

# 搜索引擎

## ——原理、技术与系统

(第二版)

李晓明 闫宏飞 王继民 著



科学出版社

(TP-5870.0101)

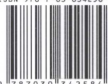
# 搜索引擎

## ——原理、技术与系统

(第二版)

[www.sciencep.com](http://www.sciencep.com)

ISBN 978-7-03-034258-4



9 787030 342584 >

定价: 48.00 元

高等教育出版中心 工科分社  
联系电话: 010-64005312  
E-mail: gk@mail.sciencep.com



# 搜索引擎

## ——原理、技术与系统

(第二版)

李晓明 闫宏飞 王继民 著

科学出版社  
北 京

## 内 容 简 介

本书系统介绍了互联网搜索引擎的工作原理、实现技术及系统构建方案。全书分三篇共13章。上篇介绍搜索引擎的基本原理和技术,讲述一个小型简单搜索引擎实现的具体细节;中篇详细讨论了大规模分布式搜索引擎系统的设计要点及其关键技术;下篇结合“中国Web信息博物馆”和“中国互联网数字资源财富库藏”的实践经验,介绍了构建大规模Web历史网页和非网页仓储系统的技术和方法,以及中文网页的自动分类与聚类、开放域问题系统的构建等。

本书层次分明,由浅入深,上篇和中篇涉及内容提供了源代码下载地址;既有深入的理论分析,也有大量的实验数据和程序,具有学习和实用双重意义。

本书可作为高等院校计算机科学与技术、软件工程、信息管理与信息系统、电子商务等专业的研究生或高年级本科生的教学参考书和技术资料;对广大从事网络技术、Web站点管理、数字图书馆、Web挖掘等研究和应用开发的科技人员有很高的参考价值;书中提供了大量源代码,除了用于构建搜索引擎之外,对于学习编程,提高编程技巧,以及实现一个大规模应用开发也有一定的参考价值。

### 图书在版编目(CIP)数据

搜索引擎:原理、技术与系统/李晓明,闫宏飞,王继民著.—2版.—北京:科学出版社,2012

ISBN 978-7-03-034258-4

I. ①搜… II. ①李…②闫…③王… III. ①互联网络-情报检索-高等学校-教材 IV. ①G354.4

中国版本图书馆CIP数据核字(2012)第090075号

责任编辑:匡敏 刘兰霞/责任校对:包志虹  
责任印制:阎磊/封面设计:速底书装

科学出版社出版

北京东黄城根北街16号

邮政编码:100717

http://www.sciencep.com

北京市农林印刷有限公司印刷

科学出版社发行 各地新华书店经销

\*

2005年4月第 一 版 开本:BS(720×1000)

2012年5月第 二 版 印张:21 3/4

2012年5月第八次印刷 字数:451 000

定价:48.00元

(如有印装质量问题,我社负责调换)

## 第二版前言

2005年4月,在华夏英才基金的支持下,本书的第一版问世。当时,互联网搜索引擎虽然在网民中已经不是一个陌生的概念,但是很少见到针对实际系统的构建,比较系统地介绍搜索引擎原理与实现技术的书。因此本书第一版的出版应该说比较及时,对不少年轻计算机技术人员起到了启蒙的作用。的确,在过去这些年里,我们多次收到读者的反馈,告知那本书对他们职业生涯的作用。最初的读者中,现在有许多都是搜索公司的骨干了。截止到2011年5月,《搜索引擎——原理、技术与系统》已经印刷了七次。一本比较专业的书,能得到那么多读者的喜欢,我们甚感欣慰。

同时我们也看到,随着互联网长盛不衰的发展,搜索的重要性日益突出;而且我们还看到一个现象,那就是相当一批有实力的互联网公司都有进入搜索领域的战略和行动,并不因为有了百度和谷歌就放弃那个市场。这种现象造成了搜索技术人才的很大缺口,尤其是有一定实际开发搜索引擎经验的人才缺口。很多高校看到这种需求,纷纷开设相关课程。许多学者看到这种需求,也纷纷写出各具特色的教材和专著。例如,郭军的《Web搜索》,董守斌和袁华的《网络信息检索》,以及刘奕群、马少平、洪涛和刘子正的《搜索引擎技术基础》。许多出版社也看到了这种需求,引进了大量相关的外版书籍。

那么,既然市场上已经有了这么多关于搜索引擎技术的书,为什么还要出版本书呢?其实这个想法我们4年前就有了,由于客观原因而耽误下来。最根本的原因是,搜索引擎技术的发展和搜索引擎技术前沿认识的深入,使得我们感到原来书中的有些内容不再重要,而有些新的内容应该包括进去。

2003年秋,在编写本书第一版的时候,主要工作基础是“天网搜索”,它曾经是中国最好的,是我们引以为自豪的搜索引擎。围绕“天网搜索”的开发,北京大学网络实验室培养了一批优秀的学生。本书第一版的内容大都是那些同学实际工作经验的总结,因此一方面总体看的确比较实用,但另一方面某些部分也不够成熟和深入。同时,过去7年来,北京大学网络实验室在搜索引擎技术方面的研究工作也有了深入进展,尤其在搜索评测与高性能索引结构方面取得了一批前沿成果,这些都是在修订中要考虑的内容。

本书保留了第一版上篇的大部分内容,即搜索引擎的基本原理,过去这么些年并没有什么变化;删除了第一版中的第九,第十二和十三章,增加了第十,第十一和十三章,分别介绍基于搜索引擎技术开发并从2002年一直运行至今的“中国Web信息博物馆”、“中国数字财富库藏”及开放域问答系统。同时,较大幅度修订了第一版中的部分小节内容。总的来看,第二版中约45%的内容是新的,且总篇幅比第一版增加

约 30%。

鉴于我们在第一版中的一个特色——详细介绍了一个小搜索引擎(TSE)并提供源代码,引起了许多读者的兴趣,纷纷下载和来邮件咨询,在此高兴地告诉读者:北京大学网络实验室将开放天网搜索系统的所有源代码。用该源代码构建的系统能搜集和处理上亿量级的网页,其体现的技术与本书中的几章内容相对应。

还有两个原因促使我们完成第二版的修订。一是 2011 年我们与百度合作承担一个国家项目“基于框计算的新一代搜索引擎与浏览器”,尽管规定的任务中没有要求,但我们认为能有这么一本书在项目完成之际面世,也是一件令人高兴的事情。二是 2003 年我们发起了“全国搜索引擎与网上信息挖掘学术研讨会”,首届在北京大学举办后,每年在全国各地由不同的高校轮流举办,很多人在会议组织工作中付出了巨大努力,他们是华南理工大学的董守斌、清华大学的李星、山东大学的马军、海南大学的雷景生、江西师范大学的王明文、大连理工大学的林鸿飞、西华大学的杜亚军,河北大学的袁方。今年已是第十届了,又回到北京大学来举办,本书的出版既是对本届会议的一个献礼,也是对过去 10 年来曾经为“全国搜索引擎与网上信息挖掘学术研讨会”做过贡献的所有朋友,以及参加会议的所有人员的感谢。

只要互联网还在发展,Web 还在发展,搜索引擎技术就会不断发展。我们相信《搜索引擎——原理、技术与系统(第二版)》能向读者更深入地展示搜索技术视野,同时也有助于启迪创新的搜索技术。同时我们愿意与读者交流,并欢迎读者指出书中难免的疏漏(lxm@pku.edu.cn; yhf@net.pku.edu.cn; wjm@pku.edu.cn; http://sewm.pku.edu.cn/book/)。

作 者

2012 年 3 月于北京大学燕园

## 第一版前言

随着互联网的不断发展和日益普及,网上的信息量在爆炸性增长,全球 Web 页面的数目已经超过 40 亿,中国的网页数目估计也超过了 3 亿。目前人们从网上获得信息的主要工具是浏览器,而通过浏览器得到信息通常有三种方式:第一,直接向浏览器输入一个关心的网址(URL),如 <http://net.pku.edu.cn>,浏览器返回所请求的网页,根据该网页内容及其包含的超链接文字(anchor text)的引导,获得自己需要的内容;第二,登录到某个知名门户网站,如 <http://www.yahoo.com>,根据该网站提供的分类目录和相关链接,逐步“冲浪”浏览,寻找自己感兴趣的东西;第三,登录到某个搜索引擎网站,如 <http://e.pku.edu.cn>,输入代表自己所关心信息的关键词或者短语,依据返回的相关信息列表、摘要和超链接引导,试探寻找自己需要的内容。

这三种方式各有特点,各有自己最适合的应用场合。第一种方式的应用是最有针对性的,例如,要了解北京大学计算机系网络与分布式系统实验室在做些什么工作,从某个渠道得知该实验室的网址为 <http://net.pku.edu.cn>,于是直接用它驱动浏览器就是最有效的方式。第二种方式的应用类似于读报,用户不一定有明确的目的,只是想看看网上有什么有意思的消息,当然这其中也可能是关心某种主题,如体育比赛、家庭生活等。第三种方式适用于用户大致知道自己要关心的内容,如“国有股减持”,但不清楚哪里能够找到相关信息(即不知道哪些 URL 能给出这样的信息);在这种场合,搜索引擎能够为用户提供一个相关内容的网址及其摘要的列表,由用户一个个试探看是否为自己需要的。现在的搜索引擎技术已经能做到在多数情况下满足用户的这种需要。CNNIC 的信息统计指出,目前搜索引擎已经成为继电子邮件之后人们用得最多的网上信息服务系统。

同时,随着网上信息资源规模的增长,尤其是其内容总体和我们社会的演化发生着越来越密切的联系,研究网上存在的海量信息逐渐成为许多学科关注的一个方向。为此,不少研究人员也有采样搜集特定内容、一定数量网页的需要。

本书以我们设计、实现并维护、运行北大“天网”搜索引擎的实际经验,介绍大规模搜索引擎的工作原理和实现技术。我们要向读者揭示,为什么向搜索引擎输入一个关键词或者短语,就能够在几秒钟内得到那么多相关的文档及其摘要,而点击其中的链接就能够被引导到文档的全文,且其中相当一部分可能正是用户需要的。

我们按照上、中、下三篇展开相关的内容。上篇讲搜索引擎的基本工作原理和技术,要解决的是为什么搜索引擎能提供如此庞大的信息查找服务这一问题,以及它在功能上有什么本质的局限性。这一篇的内容包括网页的搜集过程,网页信息的提取、组织方式和索引结构,查询提交和响应的过程以及结果产生,等等。这其中,虽然我们假定读者熟悉 URL、HTML、HTTP、CGI、MIME 等基本概念,但在上下文中也给

予了必要的介绍,力图保持行文的流畅性。这一部分内容对于需要构建小规模搜索引擎的研究人员会有直接的参考价值。

中篇讨论和大规模实用搜索引擎有关的技术问题。所谓大规模在这里指至少维护超过 1000 万的网页信息,提供相关的查询服务。所涉及的内容包括并行分布处理技术的应用,数据局部性的开发,缓存技术的应用以及搜集的网页在提供服务之前的预处理问题和高效倒排文件的建立技术等。这一部分的讨论有比较强的计算机系统结构的风格,我们将向读者展示计算机系统结构课程中的那些概念是如何生动地体现在一个实际应用系统中的。这一部分内容对构建大规模数字图书馆的技术人员也应该有帮助。

下篇介绍挑战性更强一些的内容。一般地讲,前面所述可以称为是“通用搜索引擎”,为最广泛的人群提供信息查询服务是它的基本宗旨。这意味着它的应用模式必须尽量简单,即关键词或查询短语的提交和匹配响应。尽管这已经可以解决许多问题了,但对有些重要的信息需求依然显得力不从心。例如,一个人可能会关心最近半年来网上出现了哪些关于他(她)的信息,一个企业可能要关心它做了一次大规模促销活动后一个月内网上有什么反响,一个政府机构可能会关心在一项政策法规颁布后的网上舆论。面向主题和个性化的信息查询服务就是我们试图描述的一种基本途径。这一部分内容更多地和网上中文信息处理技术有关。更准确地讲,我们要介绍网络与并行分布处理技术和中文处理技术的结合,从而实现大规模、高性能、高质量、有针对性的网上信息查询服务。这一部分内容反过来可能对从事中文信息处理的研究人员有启发作用。

本书的内容是集体智慧的结晶,主要概括了北京大学计算机科学技术系网络与分布式系统实验室自 1996 年以来的研究成果。其中许多段落直接来自同学的博士和硕士论文,他们是雷鸣、赵江华、冯是聪、单松巍、谢正茂、彭波、张志刚、龚笔宏、孟涛、咎红英等。署名作者的主要工作是把这些内容系统化,使其表述的风格统一。我们特别感谢陈葆珏教授,是她在北京大学计算机系开创了搜索引擎这一研究方向,从而使我们能在此后发扬光大,还要感谢刘建国和王建勇,是他们分别带领攻关队伍,实现了天网 1.0 和天网 2.0 版本。感谢黄蕊为本书进行的文字校对。最后,我们要感谢国家九五攻关计划、973 计划和 985 计划的支持,是它们的不断支持使我们得以将天网不断推上新的台阶,实现“让天网和中国网上信息资源规模同步成长”的理想。

作者

2004 年 5 月于北京大学燕园

# 目 录

## 第二版前言

## 第一版前言

第一章 引论	1
第一节 搜索引擎的概念	2
第二节 搜索引擎的发展历史	3
第三节 一些著名的搜索引擎	6
第四节 小结	11

## 上篇 Web 搜索引擎基本原理和技术

第二章 Web 搜索引擎工作原理和体系结构	15
第一节 基本要求	15
第二节 网页搜集	16
第三节 预处理	18
第四节 查询服务	20
第五节 体系结构	23
第六节 小结	25
第三章 Web 信息的搜集	26
第一节 概述	26
一、超文本传输协议	26
二、一个小型搜索引擎系统	27
第二节 网页搜集	30
一、定义 URL 类和 Page 类	31
二、与服务器建立连接	35
三、发送请求和接收数据	37
四、网页信息存储的天网格式	38
第三节 多道搜集程序并行工作	40
一、多线程并发工作	41
二、控制对一个站点并发搜集线程的数目	42
第四节 如何避免网页的重复搜集	43
一、记录未访问、已访问 URL 和网页内容摘要信息	43
二、域名与 IP 的对应问题	43



第五节 搜集信息的类型 .....	45
第六节 小结 .....	46
<b>第四章 对搜集信息的预处理</b> .....	47
第一节 索引网页库 .....	47
第二节 网页编码识别 .....	50
一、基本而重要的概念 .....	50
二、常用字符编码 .....	52
三、常用字符编码算法 .....	55
四、字符的输入和显示 .....	57
五、编码识别 .....	58
第三节 中文自动分词 .....	60
第四节 分析网页和建立倒排文件 .....	64
第五节 小结 .....	67
<b>第五章 信息查询服务</b> .....	68
第一节 检索的定义 .....	68
第二节 查询服务的实现 .....	69
一、结果集合的形成 .....	69
二、查询结果显示 .....	70
第三节 小结 .....	71
<b>中篇 对质量和性能的追求</b>	
<b>第六章 可扩展搜集子系统</b> .....	75
第一节 天网系统概述和集中式搜集系统结构 .....	75
一、天网系统结构 .....	75
二、集中式搜集系统 .....	76
第二节 利用并行处理技术高效搜集网页的一种方案 .....	82
一、节点间 URL 的划分策略 .....	82
二、关于性能的讨论 .....	85
三、性能测试和评价 .....	87
四、系统的动态可配置性设计 .....	90
第三节 天网分布式搜集系统 .....	92
第四节 对 Deep Web 的认识 .....	93
一、Deep Web 的成因 .....	93
二、搜索 Deep Web 的方法 .....	96
第五节 小结 .....	98
<b>第七章 网页净化与消重</b> .....	100

第一节 网页净化与元数据提取	100
一、DocView 模型	102
二、网页的表示	103
三、提取 DocView 模型要素的方法	108
四、模型应用及实验研究	112
第二节 网页消重算法	115
一、消重算法	116
二、算法评测	118
第三节 小结	121
第八章 高性能检索子系统	122
第一节 检索系统基本技术	122
一、系统设计与结构	122
二、索引创建	125
三、检索过程	127
第二节 适于查询的网页索引结构	129
一、倒排索引结构	129
二、平面位置索引	131
第三节 倒排索引压缩	135
一、倒排索引压缩技术	136
二、词典与倒排表的压缩	142
第四节 索引剪枝	150
一、静态索引剪枝方法	151
二、动态索引剪枝方法	153
第五节 混合索引技术	168
一、混合索引的原理	169
二、混合索引的实现	171
第六节 倒排文件缓存机制	173
一、倒排文件缓存	174
二、负载特性	176
三、缓存策略的选择	178
第七节 小结	178
第九章 相关排序与系统质量评估	180
第一节 传统 IR 的相关排序技术	180
第二节 链接分析与相关排序	182
一、链接分析	182
二、Web 查询模式下的新信息	184

第三节 相关排序的一种实现方案	188
一、形成网页中词项的基本权重	189
二、利用链接的结构	190
三、收集用户反馈信息	192
四、计算最终的权重	194
第四节 信息检索技术评估	195
一、信息检索技术评估指标	197
二、TREC 和 CWIRF 信息检索评估	206
三、搜索引擎技术评估	213
第五节 小结	217
下篇 Web 信息资源的组织与应用服务	
第十章 大规模 Web 历史网页仓储系统的构建	221
第一节 国外 Web 历史网页保存现状	221
一、Internet Archive	222
二、PANDORA	222
三、其他相关 Web 保存项目	223
第二节 中国 Web 信息博物馆的系统设计	224
一、Web InfoMall 的设计目标	225
二、Web InfoMall 的体系结构	225
第三节 历史网页的存储	227
一、数据的组织	228
二、存储结构	229
三、数据管理与压缩	230
四、存储性能	232
第四节 数据访问	232
一、PageID 的索引	233
二、URL 的索引	233
三、数据服务	234
四、性能与优化	235
第五节 网页的格式保存	236
第六节 小结	236
第十一章 大规模 Web 非网页信息仓储系统的构建	238
第一节 网络资源库藏相关工作	238
一、Ibiblio	239
二、Internet Archive	240

三、Wikimedia .....	240
四、中国互联网数字资源财富库藏 .....	241
第二节 CDAL 系统概况 .....	242
第三节 CDAL 系统设计 .....	244
一、系统体系结构 .....	244
二、可扩展的存储组织方案 .....	244
第四节 网络资源描述信息获取 .....	246
一、Ontology 概述 .....	247
二、描述信息获取机制 .....	247
三、改进查询的方法 .....	248
四、改进排序的方法 .....	249
第五节 基于局部聚类思想的共现词汇算法 .....	250
一、基本定义 .....	251
二、FDC 共现词汇算法 .....	251
第六节 小结 .....	252
<b>第十二章 中文网页自动分类与聚类 .....</b>	<b>253</b>
第一节 文档自动分类算法的类型 .....	253
第二节 实现中文网页自动分类的一般过程 .....	254
第三节 影响分类器性能的关键因素分析 .....	256
一、实验设置 .....	256
二、训练样本 .....	258
三、特征选取 .....	262
四、分类算法 .....	265
五、截尾算法 .....	270
六、中文网页分类器的设计方案 .....	272
第四节 天网目录导航服务 .....	272
一、问题的提出 .....	272
二、天网目录导航服务的体系结构 .....	273
三、天网目录的运行实例 .....	274
第五节 文本聚类方法 .....	275
一、文本聚类的一般过程 .....	275
二、文本间相似性的度量 .....	276
三、常用聚类算法 .....	276
四、聚类结果的评估 .....	279
五、搜索引擎返回结果的聚类 .....	280
第六节 小结 .....	281

<b>第十三章 开放域问答系统</b> .....	283
<b>第一节 概述</b> .....	283
一、问答系统的历史 .....	283
二、著名开放域问答系统介绍 .....	284
三、开放域问答系统的通用体系结构 .....	285
<b>第二节 问句的分析</b> .....	287
一、问句中的指代消解 .....	287
二、问句分类 .....	288
三、问句主题提取 .....	290
<b>第三节 文档和段落检索</b> .....	290
一、检索模型的选用 .....	291
二、查询生成 .....	291
三、查询结果排序 .....	293
四、增强索引的功能 .....	295
<b>第四节 答案提取和验证模块</b> .....	295
一、生成候选答案集合 .....	295
二、答案提取 .....	296
<b>第五节 问答系统的改进方法</b> .....	299
一、问答系统中外部资源的利用 .....	299
二、寻找特殊类问题的解决方案 .....	301
三、通过系综方法构建问答系统 .....	302
<b>第六节 问答系统的评测</b> .....	303
一、TREC 问答系统评测 .....	303
二、问答系统评测指标 .....	304
<b>第七节 实例:天网开放域问答系统</b> .....	306
<b>第八节 小结</b> .....	308
<b>参考文献</b> .....	309
<b>附录 术语</b> .....	322

## 图 表 目 录

图 1-1	2012 年 3 月在 Google 上检索“伊拉克战争”的结果 .....	2
图 1-2	2012 年 3 月在 Open Directory 上检索“伊拉克战争”的结果 .....	5
图 2-1	搜索引擎示意图 .....	15
图 2-2	搜索引擎三段式工作流程 .....	16
图 2-3	搜索引擎的体系结构 .....	23
图 3-1	TSE 搜索引擎界面 .....	28
图 3-2	TSE 查询结果页面 .....	29
图 3-3	TSE 网页快照页面 .....	29
图 3-4	TSE 系统结构 .....	30
图 3-5	Web 信息的搜集 .....	31
图 3-6	Sockets 和端口 .....	35
图 3-7	通过 Socket 建立连接 .....	36
图 4-1	网页预处理系统结构 .....	47
图 4-2	原始网页库中的记录格式 .....	48
图 4-3	索引网页库算法 .....	49
图 4-4	字符的输入和显示流程 .....	57
图 4-5	GB2312, Big5 和 GBK 字符编码分布 .....	58
图 4-6	正向减字最大匹配算法流程 .....	62
图 4-7	切词算法流程 .....	63
图 4-8	分析网页与建立倒排文件流程 .....	65
图 4-9	过滤网页中非正文信息算法 .....	65
图 4-10	正向索引表记录格式 .....	65
图 4-11	由正向索引建立反向索引 .....	66
图 5-1	信息查询的系统结构 .....	68
图 5-2	基本检索算法 .....	69
图 5-3	动态摘要算法 .....	71
图 5-4	用户查询日志的记录格式 .....	71
图 6-1	天网系统概貌 .....	76
图 6-2	搜集系统的主控结构 .....	77
图 6-3	协调进程工作算法 .....	84
图 6-4	分布式 Web 搜集系统结构 .....	85

图 6-5 负载方差 .....	88
图 6-6 并行搜集系统与集中式搜集系统的性能对比 .....	89
图 6-7 分布式系统效率 .....	89
图 6-8 URL 两阶段映射 .....	91
图 6-9 天网分布式搜集系统 P_Arthur 体系结构 .....	92
图 6-10 人才招聘网站首页 .....	94
图 7-1 用 DocView 模型提取的网页要素 .....	104
图 7-2 净化后的网页 .....	104
图 7-3 HTML Tree 结构 .....	105
图 7-4 内容块权值传递过程 .....	107
图 7-5 有主题网页 DocView 模型生成过程 .....	109
图 7-6 计算网页特征项权值的算法 .....	109
图 7-7 正文段落识别过程 .....	111
图 7-8 基于 anchor text 的超链选取算法 .....	111
图 7-9 网页净化前后分类效果对比 .....	113
图 7-10 查全率随选取关键词个数的变化 .....	120
图 8-1 检索系统集成框架结构 .....	124
图 8-2 天网 WWW 检索分布式系统构架 .....	125
图 8-3 倒排索引结构示意图 .....	129
图 8-4 按块组织的倒排链的结构 .....	130
图 8-5 位置索引的结构 .....	131
图 8-6 CLPS 结构示意图 .....	135
图 8-7 倒排链中文档号之间的 $d$ -gaps 分布图 .....	146
图 8-8 不同文档号分配下平均每个查询对应文档号序列的压缩大小 .....	146
图 8-9 不同压缩算法对文档号的解压速度 .....	147
图 8-10 不同文档号分配下平均每个查询对应词频序列的压缩大小 .....	147
图 8-11 不同压缩算法对词频的解压速度 .....	148
图 8-12 平均每个查询对应的位置信息需要的存储空间 .....	149
图 8-13 索引剪枝方法的分类 .....	151
图 8-14 MAXSCORE 算法的示例 .....	157
图 8-15 WAND 算法选择候选文档的过程 .....	159
图 8-16 基于最大块索引的支点文档号的选择示例 .....	161
图 8-17 Interval-Base 剪枝方法中文档子区间划分的示例 .....	161
图 8-18 SAAT 方法处理查询处理模式及分数累加器数量的变化 .....	164
图 8-19 当前支持高效 SR+IR 剪枝的索引结构 .....	166
图 8-20 扩展词典树结构示例 .....	172



图 8-21	扩展词典匹配查找算法 .....	173
图 8-22	搜索引擎检索系统缓存结构 .....	174
图 8-23	文档数据访问对象大小分布 .....	176
图 8-24	I/O 与 PAGE 序列序号-频度分布 .....	177
图 8-25	I/O 与 PAGE 序列时间间隔分布 .....	177
图 8-26	I/O 和 PAGE 序列中唯一模式串 .....	178
图 9-1	Inktomi 提供的几种搜索引擎技术的比较 .....	185
图 9-2	词典在系统中的地位 .....	186
图 9-3	新词学习 .....	187
图 9-4	网页的互联结构示意图 .....	191
图 9-5	信息获取技术评估的“森林” .....	197
图 9-6	查准率和召回率基础定义图示 .....	198
图 9-7	查准率和召回率例子 .....	198
图 9-8	“省事的”11 点标准召回率例子 .....	199
图 9-9	实践中召回率例子 .....	200
图 9-10	实际中的 44 个查询词的评价统计表和 $P-R$ 图 .....	202
图 9-11	测试集在检索评估中的角色 .....	208
图 9-12	帮助判断相关结果页面的计算机辅助程序入口 .....	211
图 9-13	帮助判断相关结果页面的计算机辅助程序操作界面 .....	211
图 10-1	Web InfoMall 体系结构 .....	226
图 10-2	网页数据的分割 .....	229
图 10-3	Web InfoMall 的存储结构 .....	230
图 10-4	网页的引用压缩示意图 .....	232
图 11-1	CDAL 提供的资源访问方式 .....	243
图 11-2	CDAL 系统结构图 .....	245
图 11-3	基于 Ontology 的网络资源描述信息获取 .....	248
图 11-4	概念的属性及其词汇扩展(以电影类资源为例) .....	249
图 11-5	获得描述信息的改进排序算法 .....	250
图 11-6	网络资源描述信息展示 .....	250
图 12-1	自动文档分类算法的分类 .....	254
图 12-2	中文网页自动分类的一般过程 .....	255
图 12-3	中文网页分类器的工作原理图 .....	256
图 12-4	WebSmart——一个网页实例集搜集和整理工具 .....	259
图 12-5	一种中文网页的分类体系 .....	260
图 12-6	Macro- $F_1$ 值随样本数的变化 .....	261
图 12-7	Micro- $F_1$ 值随样本数的变化 .....	261

图 12-8	CHI、IG、DF、MI 的比较(Macro- $F_1$ )	264
图 12-9	CHI、IG、DF、MI 的比较(Micro- $F_1$ )	264
图 12-10	kNN 与 NB 分类结果的比较	267
图 12-11	$k$ 的取值对分类器质量的影响(Macro- $F_1$ )	268
图 12-12	$k$ 的取值对分类器质量的影响(Micro- $F_1$ )	268
图 12-13	兰式距离法与欧式距离法对 12 个不同类别的分类情况	269
图 12-14	基于层次模型的 kNN 与基本 kNN 的比较	270
图 12-15	RCut 和 SCut 截尾算法的比较	272
图 12-16	天网目录的体系结构	274
图 12-17	天网目录导航服务	274
图 12-18	文本聚类的一般过程	275
图 12-19	层次聚类实例	277
图 12-20	$k$ -均值算法进行文本聚类的过程	278
图 12-21	搜索结果聚类系统 Carrot2	281
图 13-1	START 系统界面	285
图 13-2	Ask Jeeves 查询结果	285
图 13-3	问答系统的通用体系结构	287
图 13-4	天网开放域系统的体系结构	306
表 4-1	网页索引文件	49
表 4-2	URL 索引文件	50
表 6-1	SOIF 数据描述	78
表 6-2	SOIF 具体语法	80
表 6-3	参照序列,假设节点数为 2	87
表 7-1	类别编号对照表	113
表 7-2	消重实验结果	115
表 7-3	当 $N=10, \delta=0.01$ 时 5 种算法的查全率和准确率	119
表 7-4	考察 $\delta$ 的取值对算法 3 和 4 的影响	119
表 7-5	分段签名算法的时间复杂度及性能	120
表 7-6	基于关键词的各算法的时间复杂度及性能 ( $N=10, \delta=0.01$ )	121
表 8-1	MTF 对序列 $\langle 4, 4, 1, 4, 2 \rangle$ 进行转换的过程	142
表 8-2	对包含 100 万词条的词典使用不同编码所需要的空间	144
表 8-3	平均每个查询对应词频链的空间大小(文档号按 URL 序分配)	148
表 8-4	不同索引的组织结构及其支持的查询处理方式	155
表 8-5	数据集基本统计信息	176
表 9-1	新词学习对检索准确率的影响	188
表 9-2	影响权值的 HTML 标签	189

表 9-3	补偿因子定义表 .....	192
表 9-4	2004 中文 Web 信息检索评测提交结果 .....	210
表 9-5	主题提取 .....	212
表 9-6	导航搜索 .....	212
表 9-7	用户查询信息类别 .....	215
表 10-1	网页存储性能(个/秒) .....	232
表 10-2	网页访问性能(个/秒) .....	236
表 11-1	几个网络资源库藏系统的特征 .....	238
表 11-2	CDAL 中的资源分布 .....	243
表 12-1	样本集中类别及实例数量的分布情况表 .....	258
表 12-2	kNN 和 NB 算法的分类质量和分类效率比较 .....	267
表 12-3	欧式距离与兰式距离的比较 .....	269
表 12-4	基于层次模型的 kNN 与基本 kNN 的比较 .....	270
表 12-5	RCut 和 SCut 截尾算法的比较 .....	171
表 12-6	一个分类器的设计方案 .....	272
表 13-1	问题分类体系结构及 TREC 问答任务中问题的分布 .....	289
表 13-2	天网开放域系统在 TREC2005 中的表现 .....	307

# 第一章 引 论

信息的生产、传播、搜集与查询是人类最基本的活动之一。考虑以文字为载体的信息,传统上有图书馆、相应的编目体系和专业人员帮助我们很快找到所需的信息,其粒度通常是“书”或者“文章”。随着计算机与信息技术的发展,有了信息检索(information retrieval, IR)学科领域,有了关于图书或者文献的全文检索系统,使我们能很方便地在“关键词”的粒度上得到相关的信息。

我们注意到,上述全文检索系统一般工作在一个规模相对有限、内容相对稳定的馆藏(collection)上,被检索的对象通常是经过认真筛选和预先处理的(如人工提取出了“作者”、“标题”等元数据,形成了很好的“摘要”等),并且系统需要同时响应的查询数量通常都不会太大(如每秒钟 10 个左右)。

1994 年左右,万维网(World Wide Web,简记为 WWW 或 Web)出现。它的开放性(openness)和其上信息广泛的可访问性(accessibility)极大地鼓励了人们创作的积极性。作为一个信息源,Web 和上述全文检索系统的工作对象相比,具有许多不同的特征,它们给信息检索领域带来了新的发展机遇和技术挑战。

规模大。在短短的 10 年左右时间,人类至少生产了 40 亿网页(Google 2004),而人类有文字以来上万年里产生了大约 1 亿本书;中国网上到 2004 年初大致有了约 3 亿网页(天网 2004),而中华民族有史以来出版的书籍大约不过 275 万种。尽管书籍的容量和质量是一般网页不可比的,但在对应的时间背景上考察其文字的总体数量,我们不能不为人类在 Web 上创造文字的激情惊叹!

内容不稳定。除了不断有新的网页出现外,旧的网页也可能会因为各种原因被删除(有研究指出:50% 网页的平均生命周期大约为 50 天(Cho et al. 2000, Cho 2002))。

从原则上讲,读者数和作者数在同一个量级,形式和内容的随意性很强,权威性相对也不高,也不太可能进行人工筛选和预处理。

与生俱来的数字化、网络化。传统载体上的信息,人们目前正忙于将它们数字化、上网(花费极高),而网络信息天生如此。这个特性是一把双刃剑:一方面便于我们搜集和处理,另一方面也会使我们感到太多,蜂拥而至、鱼目混珠。

而作为要在 Web 上提供服务的信息查询系统,如搜索引擎和数字图书馆,通常要具备同时对付大量访问的能力(如每秒钟 1000 个查询),而且响应时间还要足够的快(如 1 秒钟)。

本书旨在介绍构建这类搜索引擎的有关技术。传统的 IR 是其基础,同时本书也充分讨论了由上述 Web 信息的特征所带来的新问题及其解决方案。

## 第一节 搜索引擎的概念

搜索引擎,在本书指的是一种在 Web 上应用的软件系统,它以一定的策略在 Web 上搜集和发现信息,这些信息集合对应于 Web 上一段时间内(如一周或两周)搜集的网页。对 Web 上更长时间段(如 10 年)网页的搜集和整理,我们在下篇介绍。在对信息进行处理和组织后,为用户提供 Web 信息查询服务。从使用者的角度看,这种软件系统提供一个网页界面,通过浏览器提交一个词语或者短语,可以很快返回一个可能和用户输入内容相关的信息列表(常常会是很长一个列表,如包含 1 万个条目)。这个列表中的每一条目代表一篇网页,每个条目至少有三个元素:

1) 标题:以某种方式得到的网页内容的标题。最简单的方式就是从网页的 <TITLE></TITLE> 标签中提取的内容(尽管在一些情况下并不真正反映网页的内容)。本书第七章会介绍其他形成“标题”的方法。

2) URL:该网页对应的“访问地址”。有经验的 Web 用户常常可以通过这个元素对网页内容的权威性进行判断,例如,http://www. people. com 上面的内容通常就比 http://notresponsible. net(某个假想的个人网站)上的要更权威些(不排除后者上的内容更有趣些)。

3) 摘要:以某种方式得到的网页内容的摘要。最简单的一种方式就是将网页内容的头若干字节(如前 512 字节)截取下来作为摘要。本书第七章会介绍形成“摘要”的其他方法。

通过浏览这些元素,用户对相应的网页是否真正包含他所需的信息进行判断。比较肯定的话则可以点击上述 URL,从而得到该网页的全文。图 1-1 是 2012 年 3



图 1-1 2012 年 3 月在 Google 上检索“伊拉克战争”的结果

月 15 日在 Google 搜索引擎(<http://www.google.com.hk/>)上的一个例子,用户提交了查询词“伊拉克战争”,系统返回一个相关信息列表。列表的每一条目所含内容比上述要丰富些,但核心还是那三个元素。

这个例子提示了我们一个重要的情况,即搜索引擎提供信息查询服务的时候,它面对的只是查询词。而有不同背景的人可能提交相同的查询词,关心的是和这个查询词相关的不同方面的信息,但搜索引擎通常是不知道用户背景的,因此搜索引擎既要争取不漏掉任何相关的信息,还要争取将那些“最可能被关心”的信息排在列表的前面。这也就是对搜索引擎的根本要求。除此以外,考虑到搜索引擎的应用环境是 Web,因此对大量并发用户查询的响应性能也是一个不能忽略的方面。

作为对搜索引擎工作原理的基本了解,这里有两个问题需要首先澄清。第一,当用户提交查询的时候,搜索引擎并不是即刻在 Web 上“搜索”一通,发现那些相关的网页,形成列表呈现给用户;而是事先已“搜集”了一批网页,以某种方式存放在系统中,此时的搜索只是在系统内部进行而已。第二,当用户感到返回结果列表中的某一项很可能是他需要的,从而点击 URL,获得网页全文的时候,他此时访问的则是网页的原始出处。因此,从理论上讲搜索引擎并不保证用户在返回结果列表上看到的标题和摘要内容与他点击 URL 所看到的内容一致(上面那个“伊拉克战争”的例子就是如此),甚至不保证那个网页还存在。这也是搜索引擎和传统信息检索系统的一个重要区别。这种区别源于前述 Web 信息的基本特征。为了弥补这个差别,现代搜索引擎都保存网页搜集过程中得到的网页全文,并在返回结果列表中提供“网页快照”或“历史网页”链接,保证让用户能看到和摘要信息一致的内容。

## 第二节 搜索引擎的发展历史

早在 Web 出现之前,互联网上就已经存在许多旨在让人们共享的信息资源了。那些资源当时主要存在于各种允许匿名访问的 FTP 站点(anonymous FTP),内容以学术技术报告、研究性软件居多,它们以计算机文件的形式存在,文字材料的编码通常是 PostScript 或者纯文本(那时还没有 HTML)。

为了便于人们在分散的 FTP 资源中找到所需的东西,1990 年加拿大麦吉尔大学(University of McGill)计算机学院的师生开发了一个软件,Archie。它通过定期搜集并分析 FTP 系统中存在的文件名信息,提供查找分布在各个 FTP 主机中文件的服务。Archie 能在只知道文件名的前提下,为用户找到这个文件所在的 FTP 服务器的地址。Archie 实际上是一个大型的数据库,再加上与这个大型数据库相关联的一套检索方法。该数据库中包括大量可通过 FTP 下载的文件资源的有关信息,包括这些资源的文件名、文件长度、存放该文件的计算机名及目录名等。尽管所提供服务的信息资源对象(非 HTML 文件)和本书所讨论搜索引擎的信息资源对象(HTML

网页)不一样,但基本工作方式是相同的(自动搜集分布在广域网上的信息,建立索引,提供检索服务),因此人们公认 Archie 为现代搜索引擎的鼻祖。

以 FTP 文件为对象的信息检索服务技术在 2000 年左右是比较流行的,尤其是在用户使用界面上充分采用了 Web 风格,北大天网文件检索系统就是一个例子。随着 P2P 文件共享系统的流行,更多的用户搜索文件开始使用 P2P 客户端,FTP 搜索逐渐淡出。近年来,随着智能手机和平板电脑的流行,搜索已经不限于在电脑上完成,而且有些应用天生适合移动设备的小屏幕,如微博、微信和飞信等;搜索的技术更多地不同的领域得到应用,如淘宝网(<http://www.taobao.com/>)中提供的检索、推荐和过滤功能;搜索的对象也不限于网页和文件,如微博、评论、标签、视频、音频、事件和图片等。鉴于本书写作定位的关系,后面将主要讨论网页搜索引擎的相关问题。

以 Web 网页为对象的搜索引擎和以 FTP 文件为对象的检索系统一个基本的不同点在于搜集信息的过程。前者是利用 HTML 文档之间的链接关系,在 Web 上一个网页一个网页地“爬取”(crawl),将那些网页“抓”(fetch)到本地后进行分析;后者则是根据已有的关于 FTP 站点地址的知识(如得到了一个站点地址列表),对那些站点进行访问,获得其文件目录信息,并不真正将那些文件下载到系统上来。因此,如何在 Web 上“爬取”,就是搜索引擎要解决的一个基本问题。在这方面,1993 年 Matthew Gray 开发了 World Wide Web Wanderer,它是世界上第一个利用 HTML 网页之间的链接关系来监测 Web 发展规模的“机器人”(robot)程序。刚开始时它只用来统计互联网上的服务器数量,后来则发展为能够通过它检索网站域名。鉴于其在 Web 上沿超链“爬行”的工作方式,这种程序有时也称为“蜘蛛”(spider)。因此,在文献中 crawler、spider、robot 一般都指的是相同的事物,即在 Web 上依照网页之间的超链关系一个个抓取网页的程序,通常也称为“搜集”。在搜索引擎系统中,也称为网页搜集子系统。

现代搜索引擎的思路源于 Wanderer,不少人在 Matthew Gray 工作的基础上对它的蜘蛛程序做了改进。1994 年 7 月,Michael Mauldin 将 John Leavitt 的蜘蛛程序接入到其索引程序中,创建了大家现在熟知的 Lycos,成为第一个现代意义的搜索引擎。在那之后,随着 Web 上信息的爆炸性增长,搜索引擎的应用价值也越来越高,不断有更新、更强的搜索引擎系统推出。这其中,特别引人注目的是 Google,虽然是个姗姗来迟者(1998 年才推出),但由于其采用了独特的 PageRank 技术,使它很快后来居上,成为当前全球最受欢迎的搜索引擎(作者 2003 年初访问印度,就听到总统阿卜杜勒·卡拉姆讲他经常用 Google 在网上查找信息)。

在中国,据我们所知,对搜索引擎的研究起源于“中国教育科研网”(CERNET)一期工程中的子项目,北京大学计算机系的项目组在陈葆珏教授的主持下于 1997 年 10 月在 CERNET 上推出了天网搜索 1.0 版本。该系统在这几年里不断发展,目前已成为中国最大的公益性搜索引擎(<http://e.pku.edu.cn>)。在这之后,几位在美国



留学的华人学者回国创业,成立了百度公司,于2000年推出了“百度”商业搜索引擎(<http://www.baidu.com>),并一直处于国内搜索引擎的领先地位。我们看到搜狐公司也在中国推出了一个大规模搜索引擎(<http://www.sogou.com>),用起来感觉也不错,但往后发展如何,还有待时间的考验。

当我们谈及搜索引擎的时候,不应该忽略另外一个几乎是同期发展出来的事物:基于目录的信息服务网站。1994年4月,斯坦福(Stanford)大学的两名博士生,David Filo 和 杨致远(Gerry Yang)共同创办了Yahoo! 门户网站,并成功地使网络信息搜索的概念深入人心。1996年中国出现了类似的网站,“搜狐”(http://www.sohu.com)。在许多场合,也称Yahoo! 之类的门户网站提供的信息查找功能为搜索引擎。但从技术上讲,这样的门户中提供的搜索服务和前述搜索引擎是很不同的。而且目前这类门户网站通常采用两种方式的结合来提供服务,即人工编辑结合自动搜索。这样的门户依赖的是人工整理的网站分类目录,一方面,用户可以直接沿着目录导航,定位到他所关心的信息;另一方面,用户也可以提交查询词,让系统将他直接引导到和该查询词最匹配的网站。图1-2就是我们在Open Directory Project([The screenshot shows the Open Directory Project search results for the query '伊拉克战争' \(Iraq War\). The page layout includes a search bar at the top with the text 'open directory project' and 'Aol Search'. Below the search bar, the results are organized into two main sections: 'Open Directory Categories \(1-1 of 1\)' and 'Open Directory Sites \(1-8 of 8\)'. The 'Open Directory Categories' section lists 'World Chinese Simplified' with a sub-category '社会、专题、战争与冲突: 伊拉克战争 \(0\)'. The 'Open Directory Sites' section lists eight websites, each with a brief description and a URL. The websites include 'World Chinese Simplified', 'Iraq War', 'Iraq War News', 'Iraq War News', 'Iraq War News', 'Iraq War News', 'Iraq War News', and 'Iraq War News'. At the bottom of the page, there is a search bar with the text '伊拉克战争' and a 'New Search' button. The footer of the page contains the text 'Copyright © 2012 Website'.](http://</a></p>
</div>
<div data-bbox=)

图 1-2 2012 年 3 月在 Open Directory 上检索“伊拉克战争”的结果

www.dmoz.org/)上查询“伊拉克战争”的结果。一般来讲,前者的信息搜索会更全面些,后者则会准确些。在没有特殊说明的情况下,本书中所讨论的“搜索引擎”不包括 Yahoo! 和搜狐这样的搜索方式。

随着网上信息越来越多,单纯靠人工整理网站目录取得较高精度查询结果的优势逐渐退化——对海量的信息进行高质量的人工分类已经不太现实。目前有两个发展方向。一是利用文本自动分类技术,在搜索引擎上提供对每篇网页的自动分类,这方面最先看到的例子是 Google 的“网页分类”选项,但它分类的对象只是英文网页。在中文方面,文本自动分类的研究工作有很多,但我们知道的第一个在网上提供较大规模网页自动分类服务的是北大网络实验室冯是聪和龚笔宏等人的工作(冯是聪 2003),他们于 2002 年 10 月在天网搜索上挂接了一个 300 万网页的分类目录。另一个发展方向是将自动网页爬取和一定的人工分类目录相结合,希望形成一个既有高信息覆盖率,也有高查询准确性的服务。

互联网上信息量在不断增加,信息的种类也在不断增加。例如,除了我们前面提到的网页和文件,还有微博、论坛、专业数据库等。同时上网的人数也在不断增加,网民的成分也在发生变化。一个搜索引擎要覆盖所有的网上信息查找需求已出现困难,因此各种主题搜索引擎、个性化搜索引擎、问答式搜索引擎等纷纷兴起。这些搜索引擎虽然还没有实现如通用搜索引擎那样的大规模应用,但随着互联网的发展,我们相信它们的生命力会越来越旺盛。另外,即使通用搜索引擎的运行现在也开始出现分工协作,有了专业的搜索引擎技术和搜索数据库服务提供商。如美国的 Inkto-mi,它本身并不是直接面向用户的搜索引擎,但向包括 Overture(原 GoTo)、LookSmart、MSN、HotBot 等在内的其他搜索引擎提供全文网页搜集服务。从这个意义上说,它是搜索引擎数据的来源。

搜索引擎出现虽然只有 20 年左右的历史,但在 Web 上已经有了确定不移的地位。据 CNNIC 统计,它已经成为继电子邮件之后的第二大 Web 应用。虽然它的基本工作原理已经相当稳定,但在其质量、性能和服务方式等方面的提高空间依然很大,研究成果层出不穷,是每年 WWW 学术年会<sup>①</sup>的重要论题之一。

### 第三节 一些著名的搜索引擎

为了让感兴趣的读者有目的地试一试,我们整理了一些当前主流的搜索引擎,包括网址、首页面图片及其介绍。这些搜索引擎提供多语言的支持,可以满足不同母语读者的需求。

<sup>①</sup> International WWW Conference Committee, 网址 <http://www.iw3c2.org>.

Google, <http://www.google.com>



Google 作为在网络搜索页面的首选是无愧于这个称号的。它基于自动爬取网页的搜集器的服务既保证了能够覆盖广泛的网页,同时在查询效果上也表现得极其优秀。

通过其首页面的标签,可以方便地检索网络上的网页、图像、视频、论坛和新闻。它提供网页快照,保证在即使存有网页的服务器暂时出现故障时仍可浏览该网页的内容,或者可以浏览到不是最新版的该网页的内容;拼写检查,如果您的查询词包含错误的拼写,它会提示正确的查询词;股票行情查询;街区地图查询等特殊功能。此外,Google 工具条因为提供了方便存取 Google 和它的特性而为其赢得了一定的声誉。

Google 除了提供无需付费的排序结果,还有自己的竞价排名程序。与其他提供此项服务的公司一样,依据点击才有花费,竞价排名程序在 Google 的返回结果中放置广告。Google 还提供自己的无需付费的排序结果给其他一些搜索引擎。

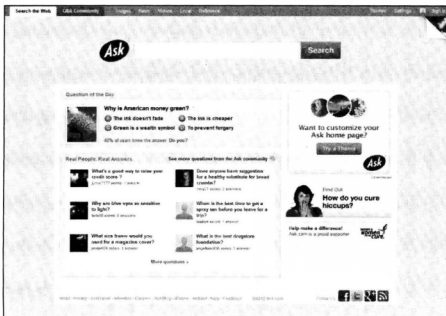
Google 最初起源于斯坦福大学的 BackRub 项目,当时是由学生 Larry Page 和 Sergey Brin 主要负责。到了 1998 年,BackRub 更名为 Google,并且走出校园成为一个公司。

**Yahoo! Search, <http://search.yahoo.com>**



Yahoo! Search 的前身是 AllTheWeb, 作为一个优秀的基于搜集器的搜索引擎, AllTheWeb 提供广泛的网络覆盖与显著的相关性。除了提供网页查询, AllTheWeb 还提供新闻、图像、视频和音频的检索。AllTheWeb 于 1999 年 5 月推出, 先是由 FAST 运作; 2003 年 4 月 Overture 收购了 AllTheWeb; 后来 Yahoo! 买下了 Overture, 现在的 AllTheWeb 由 Yahoo! 运作。

**Ask Jeeves, <http://www.askjeeves.com>**



Ask Jeeves 最初获得关注是在 1998 年和 1999 年。作为自然语言搜索引擎,能够让用户通过输入问题来得到查询结果,并且所得到的结果看起来还可以接受。

事实上,技术并不是 Ask Jeeves 运行很好的原因。在幕后,公司曾经指定 100 个编辑人员监视查询日志。然后这 100 个人上网查找与最常用查询词最相关的网页链接。目前,Ask Jeeves 仍然在使用人来参与结果的查找,但是现在的编辑只有 10 个人左右。尽管如此,通过人的参与提供答案仍然是一个卖点,尤其对于那些新接触网络的人,他们会想使用 Ask Jeeves。对于通常的查询,人工选择的匹配结果让人感觉非常的相关。如果有人工答案,这些结果出现在查询结果页面的最上端。除了人工参与外,Ask Jeeves 还利用基于搜集器的技术提供查询结果给用户。这些结果来自它所拥有的 Teoma 搜索引擎。

Vivisimo, <http://www.vivisimo.com>

The screenshot shows the Vivisimo search engine interface. At the top, there is a navigation bar with links: company | products | solutions | converters | names | partners | press. Below this is a search bar with the text "北大天网" and a "Search" button. To the right of the search bar, there are links for "the Web", "Advanced", and "Help". Below the search bar, there is a section for "Refer us to a friend" and a link for "NEW Toolbar or Monitor".

The main content area is divided into two columns. The left column lists search results with titles like "北大天网 (110)", "搜索引擎, 北大天网搜索引擎 (43)", "北大天网搜索引擎 (5)", "天网TP引擎 北大天网\_高级搜索 (4)", "百度mp3搜索 北大天网搜索引擎 (6)", "搜索引擎, 国内外mp3搜索引擎 部分新到资源 (5)", "北大天网 网县中文 河南搜索 (5)", "英文分库 gnu 整合 北大天网 搜索引擎 (7)", "梦游理工大全文档 北大天网 (7)", "百度搜索 (4)", "北大天网 易搜索 (2)".

The right column shows a list of search results with titles like "1. 搜索引擎北大天网中英文搜索引擎 [new window] [frame] [preview]", "2. 北大天网搜索引擎高级搜索技巧 [new window] [frame] [preview]", "3. 北大在线搜索 [new window] [frame] [preview]", "4. 搜索引擎大集中,fp搜索引擎,mp3搜索,mp3音乐搜索 www.haxia.com [new window] [frame] [preview]", "5. latina.latina.latina [new window] [frame] [preview]".

At the bottom, there is a section for "Find in clusters:" and a search bar for "Enter Keywords".

Vivisimo 于 2000 年 6 月由卡耐基-梅隆大学(CMU)推出,作为不同于基于搜集器的元搜索引擎,它有自己的独到之处。它把其他搜索引擎的返回结果利用自动聚类的办法来满足不同类型客户的需要。在搜索引擎上,任何人搜索同一个词的结果都是一样。这样明显不能满足访问者。科学家搜索“星球”,可能是希望了解星球的知识,但普通人可能是想找“星球大战”电影,但搜索引擎所给的都是一样的结果。如何满足这些不同类型的访问者,需要对搜索结果进行个性化处理。搜索结果排序从单一化到个性化,Vivisimo 已经迈出了一步。

现在的 Vivisimo 主营业务是信息优化。

**Bing**, <http://cn.bing.com>

Bing 是微软于 2009 年 5 月推出的,在功能上与前面对 Google 的描述类似。



**Baidu(百度)**, <http://www.baidu.com>



百度于 2000 年推出,是目前在中国最成功的一个商业搜索引擎,主要提供中文信息检索,并且为门户网站提供搜索结果服务。搜索范围涵盖了大陆(包括香港、台湾、澳门)、新加坡等华语站点及北美、欧洲的部分站点。拥有的中文信息总量达到 1000 亿,每天的网页更新量有 50 亿以上。

Tianwang(天网), <http://e.pku.edu.cn>

天网于1997年10月开始提供服务,是中国最早的搜索引擎。它由北京大学网络与分布式系统实验室开发并维护运行,搜集了中国范围内大量的网络信息资源,尤其是较全面地覆盖了中国教育科研网(CERNET)内的资源。天网目前搜索的信息资源除已经超过3亿的网页外,还包括2000多万各种非网页类型的文件,是目前世界上最大的中文搜索引擎之一。目前由于商用搜索引擎更新快、内容覆盖广,天网搜索已经暂停更新自己的索引,而是利用自己的技术积累优势,开展相关的信息检索关键技术研究。

本书所介绍的技术内容主要就是以天网为背景展开的。



## 第四节 小 结

随着 Web 的发展,网页数量的激增,在海量数据中找到用户所需信息,使搜索引擎毋庸置疑成为 Web 上最重要的应用之一。本章首先给出了搜索引擎的概念及特点,然后介绍搜索引擎的发展历史,最后提供了当前一些主流搜索引擎的网址和介绍。





# 上篇 Web 搜索引擎基本原理和技术

上篇的主要目的是向读者介绍典型 Web 搜索引擎的基本工作原理,并通过一个实例具体展示该工作原理中各个环节的一种实现方法,以期使读者很快从技术上对搜索引擎系统的全貌有一个透彻的了解。

我们首先指出,所谓“搜索引擎”,说到底是一个计算机应用软件系统,或者说是一个网络应用软件系统。从网络用户的角度看,它根据用户提交的类自然语言查询词或者短语,返回一系列很可能与该查询相关的网页信息,供用户进一步判断和选取。为了有效地做到这一点,它大致上被分成三个功能模块,或者三个子系统,即网页搜集、预处理和查询服务。第二章详细分析了这三个部分的主要功能和其中需要关注的种种问题。应该指出,在实践中这三个部分是相对独立的,它们的工作形成了搜索引擎工作的三个阶段,通常分别都由人工启动。同时我们注意到,在早期的搜索引擎中,系统处理的网页数量少,预处理部分的工作比较简单,只是涉及汉语的分词(英文还没有这个问题)和建索引,因此也有将分词合并到网页搜集过程中,将建索引归到查询服务子系统中,从而整个系统看起来只有两个模块的安排<sup>①</sup>。

在介绍了基本原理后,第三、四、五章分别就上述三个阶段中的技术要求给出了一种实现方案。为了便于读者对搜索引擎技术在短时间内有一个全面实在的掌握,这三章的基本写作风格是提出问题,给出解决思路,然后是对应程序实现要点的注释和讲解。如果要掌握每一个细节(如需要开发一个搜索引擎),要求读者对 C++ 程序设计语言比较熟悉。然而,从了解现代搜索引擎技术原理的需求来看,中篇和下篇并不很依赖这

---

<sup>①</sup> 1997 年 10 月我们在 CERNET 上发布的天网 1.0 版本就是这种结构,每抓来一个网页就立即在内存分词,然后将得到的结果存入数据库中,供建索引程序直接使用。

三章的内容,因此对 C++ 程序设计语言不熟的读者可以跳过这三章,直接阅读中篇和下篇,或者不一定要一句句探究那些程序段落的逻辑。程序代码可以在(Tse 2004)下载。

对于希望动手构建搜索引擎的读者来说,掌握了这一篇的内容,直接用我们提供的实例代码,应该能够很快(如一周)构建出一个可用的小型通用搜索引擎。同时我们指出,一个实用的大规模搜索引擎还有许多其他重要问题要解决,集中在效率和质量两个方面,我们安排在中篇讨论。

## 第二章 Web 搜索引擎工作原理和体系结构

本章介绍搜索引擎的基本工作原理和它作为一种网络应用软件的体系结构。在第三、四、五章中,我们将以一个实际的例子,具体展开在这些原理基础上实现的一种方案。通过这几章,读者将得到一个可实际运行搜索引擎的实现细节。

### 第一节 基本要求

如在第一章第二节所述,搜索引擎是一个网络应用软件系统,如图 2-1 所示,对它有如下基本要求。

能够接受用户通过浏览器提交的查询词或者短语,记作  $q$ ,例如,“非典”、“伊拉克战争”、“床前明月光”等。

在一个可以接受的时间内返回一个和该用户查询匹配的网页信息列表,记作  $L$ 。第一章讲过,这个列表的每一条目至少包含三个元素(标题、网址链接、摘要)。

这里有几个问题需要注意,它们对应上面黑体的文字:

“可以接受的时间”,也就是响应时间。对于在 Web 上面向广大用户提供服务的软件来说,这个时间不能太长,通常也就在“秒”这个量级。这是衡量搜索引擎可用性的一个基本指标,也是和传统信息检索系统的一个差别。更进一步的,这样的响应时间要求不仅要能满足单个用户查询,而且要能在系统设计负载的情况下满足所有的用户。也就是说,系统应该在额定吞吐率的情况下保证秒级响应时间。这其中详细的分析将在中篇第八章展开。

“匹配”,指的是网页中以某种形式包含有  $q$  的内容,其中最简单、最常见的形式就是  $q$  在其中直接出现。不过后面我们会看到,如果一个搜索引擎就是以百分之百满足这种简单的包含关系为目标,即使实现了也并不就达到了最好的效果。

“列表”,这蕴含着一种“序”(rank)。在绝大多数情况下, $L$  是相当长的,如超过 1 万个条目(这是和图书馆全文检索系统的又一个不同,那里返回的列表通常较短,如几十个条目)。这不仅是由于 Web 上的信息量大,也由于搜索引擎的查询方式简单。简单,意味着抽象;抽象,意味着有更多的具体事物可能是它的体现。对于一个长长的列表,很少有用户有耐心都审视一遍(不仅是因为长,还因为大多数使用搜索引擎的用户通常都是“找到为止”,而不是“不全部找到不罢休”,加上这个列表中一个

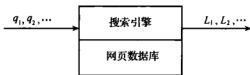


图 2-1 搜索引擎示意图

用户关心的其实只占很少的比例)。有分析统计表明,用户平均察看返回结果不超过 2 页(Baldi et al. 2003, Wang et al. 2001, 单松巍 2003)。

现代大规模高质量搜索引擎一般采用如图 2-2 所示的称之为三段式的工作流程,即:网页搜集、预处理和查询服务。



图 2-2 搜索引擎三段式工作流程

## 第二节 网页搜集

搜索引擎这样一个软件系统应该是何种工作方式？如果说软件系统是工作在某个数据集合上的程序的话,这个软件系统操作的数据不仅包括内容不可预测的用户查询,还要包括在数量上动态变化的海量网页,并且这些网页不会主动送到系统来,而是需要由系统去抓取。

首先,我们考虑抓取的时机:事先还是即时。我们都有经验,在网络比较畅通的情况下,从网上下载一篇网页大约需要 1 秒钟左右,因此如果在用户查询的时候即时去网上抓来成千上万的网页,一个个分析处理,和用户的查询匹配,不可能满足搜索引擎的响应时间要求。不仅如此,这样做的系统效益也不高(会重复抓取太多的网页);面对大量的用户查询,不可能想象每来一个查询,系统就到网上“搜索”一次。

因此我们看到,大规模搜索引擎服务的基础应该是一批预先搜集好的网页(直接或者间接<sup>①</sup>)。这一批网页如何维护?可以有两种基本的考虑。

定期搜集,每次搜集替换上一次的内容,我们称之为“批量搜集”。由于每次都是重新来一次,对于大规模搜索引擎来说,每次搜集的时间通常会花几周。而由于这样做开销较大,通常两次搜集的间隔时间也不会很短(如早期天网的版本大约每 3 个月来一次,Google 在一段时间曾是每隔 28 天来一次)。这样做的好处是系统实现比较简单,主要缺点是“时新性”(freshness)不高,还有重复搜集所带来的额外带宽的消耗。

增量搜集,开始时搜集一批,往后只是:①搜集新出现的网页;②搜集那些在上次搜集后有过改变的网页;③发现自从上次搜集后已经不再存在了的网页,并从库中删除。由于除新闻网站外,许多网页的内容变化并不是很经常的(有研究指出 50%网页的平均生命周期大约为 50 天(Cho et al. 2000, Cho 2002)),这样做每次搜集的网页量不会很大(例如,我们在 2003 年初估计中国每天有 30 万~50 万变化了的网

<sup>①</sup> 所谓“间接”,指的是提供搜索服务的系统可能利用别人已经事先抓好的数据,元搜索引擎就是如此。

页),于是可以经常启动搜集过程(如每天)。30 万网页,一台 PC 机,在一般的网络条件下,半天也就搜集完了。这样的系统表现出来的信息时新性就会比较高,主要缺点是系统实现比较复杂;这种复杂还不仅在于搜集过程,而是还在于下面要谈到的建索引的过程。

上面讲的是系统网页数据库维护的基本策略。在这两种极端的情况之间也可能有一些折中的方案,J. Cho 博士在这方面做过深入的研究(Cho et al. 2000, Cho 2002),他根据一种网页变化模型和系统所含内容时新性的定义,提出了相应优化的网页搜集策略。其中一个有趣的结论是:在系统搜集能力一定的情况下,若有两类网页(如“商业”和“教育”),它们的更新周期差别很大(如“商业”类网页平均更新周期是“天”,而“教育”类网页平均更新周期是“月”),则系统应该将注意力放在更新慢的网页上(Cho et al. 2000),以使系统整体的时新性达到比较高的取值。

在具体搜集过程中,如何抓取一篇篇的网页,也可以有不同的考虑。最常见的一种是所谓“爬取”:将 Web 上的网页集合看成是一个有向图,搜集过程从给定起始 URL 集合  $S$ (或者说“种子”)开始,沿着网页中的链接,按照先深、先宽或者某种别的策略遍历,不停地从  $S$  中移除 URL,下载相应的网页,解析出网页中的超链接 URL,看是否已经被访问过,将未访问过的那些 URL 加入集合  $S$ 。整个过程可以形象地想象为一个蜘蛛(spider)在蜘蛛网(Web)上爬行(crawl)。后面我们会看到,真正的系统其实是多个“蜘蛛”同时在爬。

这种方式的好处除了概念很漂亮,一般实现起来也不困难外,还有一条很重要的一条是容易通过一定的策略,使搜集到的网页相对比较“重要”。前面提过,任何搜索引擎是不可能将 Web 上的网页搜集完全的,通常都是在其他条件的限制下决定搜集过程的结束(如磁盘满,或者搜集时间已经太长了)。因此就有一个尽量使搜到的网页比较重要的问题,这对于那些并不追求很大的数量覆盖率的搜索引擎特别重要。研究表明(Najork et al. 2001),按照先宽搜索方式得到的网页集合要比先深搜索得到的集合重要(这里当然有一个重要性的指标问题)。这种方式的一个困难是要从每一篇网页中提取出所含的 URL。由于 HTML 的灵活性,其中出现 URL 的方式各种各样,将这个环节做得彻底并不容易(例如,我们现在还没有很好的简单办法从 JavaScript 脚本中提取 URL)。同时,由于 Web 的“蝴蝶结”形状(Broder et al. 2000),这种方式搜集到的网页不大会超过所有目标网页<sup>①</sup>数量的 2/3。

另外一种可能的方式是在第一次全面网页搜集后,系统维护相应的 URL 集合  $S$ ,往后的搜集直接基于这个集合。每搜到一个网页,如果它发生变化并含有新的 URL,则将它们对应的网页也抓回来,并将这些新 URL 也放到集合  $S$  中;如果  $S$  中某个 URL 对应的网页不存在了,则将它从  $S$  中删除。这种方式也可以看成是一种极端的先宽搜索,即第一层是一个很大的集合,往下最多只延伸一层。

① 所谓“目标网页”指的是搜索引擎设计覆盖的网页范围。例如,Google 覆盖全球,天网覆盖全中国。

还有一种方法是让网站拥有者主动向搜索引擎提交它们的网址(为了宣传自己,通常会有这种积极性),系统在一定时间内(两天到数月不等)定向向那些网站派出“蜘蛛”程序,扫描该网站的所有网页并将有关信息存入数据库中。大型商业搜索引擎一般都提供这种功能。

### 第三节 预 处 理

得到海量的原始网页集合,离面向网络用户的检索服务之间还有相当的距离。宏观地看,服务子系统是一个程序。采用 Wirth 关于“程序 = 算法 + 数据结构”的观点来考察这个程序,一个合适的数据结构是查询子系统工作的核心和关键。这里只是指出:现行最有效的数据结构是“倒排文件”(inverted file);倒排文件是用文档中所含关键词作为索引,文档作为索引目标的一种结构(类似于普通书籍中,索引是关键词,书的页面是索引目标)。我们在第八章中有进一步分析。下面讨论从网页集合形成这样的倒排文件过程中的几个主要问题,即我们所说的“预处理”。主要包括四个方面:关键词的提取,“镜像网页”(网页的内容完全相同,未加任何修改)或“转载网页”(near-replicas,主题内容基本相同但可能有一些额外的编辑信息等,转载网页也称为“近似镜像网页”)的消除,链接分析和网页重要程度的计算。

#### 1. 关键词的提取

随便取一篇网页的源文件(如通过浏览器的“查看源文件”功能),我们可以看到其中的情况纷乱繁杂。除了我们从浏览器中能够正常看到的文字内容外,还有大量的 HTML 标记。根据天网统计,网页文档源文件的大小(字节量)通常大约是其中内容大小的 4 倍(如 <http://net.pku.edu.cn> 就是如此)。另外,由于 HTML 文档产生来源的多样性,许多网页在内容上比较随意,不仅文字不讲究规范、完整,而且还可能包含许多和主要内容无关的信息(如广告,导航条,版权说明等)。这些情况既给有效的信息查询带来了挑战,也带来了一些新的机遇,在后面的章节将会有进一步的论述。这里我们只是指出,为了支持后面的查询服务,需要从网页源文件中提取出能够代表它的内容的一些特征。从人们现在的认识和实践来看,所含的关键词即为这种特征最好的代表。于是,作为预处理阶段的一个基本任务,就是要提取出网页源文件的内容部分所含的关键词。对于中文来说,就是要根据一个词典  $\Sigma$ , 用一个所谓“切词软件”,从网页文字中切出  $\Sigma$  所含的词语来。在那之后,一篇网页主要就由一组词来近似代表了,  $p = \{t_1, t_2, \dots, t_n\}$ 。一般来讲,我们可能得到很多词,同一个词可能在一篇网页中多次出现。从效果(effectiveness)和效率(efficiency)考虑,不应该让所有的词都出现在网页的表示中,要去掉诸如“的”,“在”等没有内容指示意义的词,称为“停用词”(stop word)。这样,对一篇网页来说,有效的词语数量在 200 左右。

## 2. 重复或转载网页的消除

与生俱来的数字化和网络化给网页的复制以及转载和修改再发表带来了便利,因此我们看到 Web 上的信息存在大量的重复现象。天网在 2003 年的一次大规模统计分析表明,网页的重复率平均大约为 4。也就是说,当你通过一个 URL 在网上看到一篇网页的时候,平均还有另外 3 个不同的 URL 也给出相同或者基本相似的内容。这种现象对于广大的网民来说是有正面意义的,因为有了更多的信息访问机会。但对于搜索引擎来说,则主要是负面的,它不仅在搜集网页时要消耗机器时间和网络带宽资源,而且如果在查询结果中出现,无意义地消耗了计算机显示屏资源,也会引来用户的抱怨,“这么多重复的,给我一个就够了”。因此,消除内容重复或主题重复的网页是预处理阶段的一个重要任务。第七章对此有详细的分析论述。

## 3. 链接分析

前面提到,大量的 HTML 标记既给网页的预处理造成了一些麻烦,也带来了一些新的机遇。从信息检索的角度讲,如果系统面对的仅仅是内容的文字,我们能依据的就是“共有词汇假设”(shared bag of words),即内容所包含的关键词集合,最多加上词频(term frequency 或  $tf$ 、 $TF$ )和词在文档集中出现的文档频率(document frequency 或  $df$ 、 $DF$ )之类的统计量。而  $TF$  和  $DF$  这样的频率信息能在一定程度上指示词语在一篇文档中的相对重要性或者和某些内容的相关性,这是很有意义的。有了 HTML 标记后,情况还可能进一步改善,例如,在同一篇文档中,  $\langle H1 \rangle$  和  $\langle /H1 \rangle$  之间的信息很可能就比在  $\langle H4 \rangle$  和  $\langle /H4 \rangle$  之间的信息更重要。特别地,HTML 文档中所含的指向其他文档的链接信息是人们近几年来特别关注的对象,认为它们不仅给出了网页之间的关系,而且还对判断网页的内容有很重要的作用。例如,“北大学报”这几个字在北京大学学报社会科学版的主页上是没的,因此一个仅靠内容文字分析的搜索引擎就不可能返回该主页作为结果。但是北京大学主页上是用“北大学报(社)”作为链接信息指向了北京大学学报社会科学版的主页。因此在很好利用链接信息的搜索引擎中应该能返回北京大学学报社会科学版的主页。

## 4. 网页重要程度的计算

搜索引擎返回给用户的,是一个和用户查询相关的结果列表。列表中条目的顺序是很重要的一个问题。由于面对各种各样的用户,加之查询的自然语言风格,对同样的  $q_0$  返回相同的列表肯定是不能使所有提交  $q_0$  的用户都满意的(或者都达到最高的满意度)。因此搜索引擎实际上追求的是一种统计意义上的满意。人们认为 Google 目前比天网好,是因为在多数情况下前者返回的内容要更符合用户的需要,而不是所有情况下都如此。如何对查询结果进行排序有很多因素需要考虑,后面将有深入的讨论。这里只是概要解释在预处理阶段可能形成的所谓“重要性”因素。顾



名思义,既然是在预处理阶段形成的,就是和用户查询无关的。如何讲一篇网页比另外一篇网页重要?人们参照科技文献重要性的评估方式,核心想法就是“被引用多的就是重要的”。“引用”这个概念恰好可以通过 HTML 超链在网页之间体现得非常好,作为 Google 创立核心技术的 PageRank 就是这种思路的成功体现(Page et al. 1998)。除此以外,人们还注意到网页和文献的不同特点,即一些网页主要是大量对外的链接,其本身基本没有一个明确的主题内容,而另外有些网页则被大量的其他网页链接。从某种意义上讲,这形成了一种对偶的关系,这种关系使得人们可以在网页上建立另外一种重要性指标(Kleinberg 1998)。这些指标有的可以在预处理阶段计算,有的则要在查询阶段计算,但都是作为在查询服务阶段最终形成结果排序的部分参数。

## 第四节 查询服务

如上所述,从一个原始网页集合  $S$  开始,预处理过程得到的是对  $S$  的一个子集的元素的一种内部表示,这种表示构成了查询服务的直接基础。对每个元素来说,这种表示至少包含如下几个方面:

- 原始网页文档;
- URL 和标题;
- 编号;
- 所含的重要关键词的集合(以及它们在文档中出现的位置信息);
- 其他一些指标(如重要程度、分类代码等)。

而系统关键词总体的集合和文档的编号一起构成了一个倒排文件结构,使得一旦得到一个关键词输入,系统能迅速给出相关文档编号的集合输出。

然而,如同我们在第一章提到的,用户通过搜索引擎看到的不是一个“集合”,而是一个“列表”。如何从集合生成一个列表,是服务子系统的主要工作。从搜索引擎系统功能划分的角度,有时候将倒排文件的生成也作为服务子系统的一部分功能,但我们这里将它划分到预处理阶段中觉得更方便些。换句话讲,服务子系统是在服务进行的过程中涉及的相关软件程序,而为这些软件程序事先准备数据的程序都算在预处理子系统中。下面来看对服务子系统的要求和其工作原理,主要有三个方面。

### 1. 查询方式和匹配

查询方式指的是系统允许用户提交查询的形式。考虑到各种用户的不同背景和不同的信息需求,不可能有一种普遍适用的方式。一般认为,对于普通网络用户来说,最自然的方式就是“要什么就输入什么”。但这是一种相当模糊的说法。例如,用户输入“北京大学”,可能是他想了解北京大学目前有些什么信息向外发布,如想看看今年的招生政策(于是希望看的是北大网站上的内容);也可能是他想了解外界目前

对北京大学有什么评价(于是希望看到的是其他权威网站上关于北大的消息)。这是两种相当不同的需求。在其他一些情况下,用户可能关心的是间接信息,例如,“喜马拉雅山的高度”,8848 应该他需要的,但不可能包含在这短语中。而用户输入“惊起一滩鸥鹭”则很可能是想知道该词的作者是谁,或者希望能提醒前面几句是什么。尽管如此,用一个词或者短语来直接表达信息需求,希望网页中含有该词或者该短语中的词,依然是主流的搜索引擎查询模式。这不仅是因为它的确代表了大多数的情况,还因为它比较容易实现。这样,一般来讲,系统面对的是查询短语。就英文来说,它是一个词的序列;就中文来说,它是包含若干个词的一段文字。一般地,我们用  $q_0$  表示用户提交的原始查询,例如,  $q_0 = \text{“网络与分布式系统实验室”}$ 。它首先需要被“切词”(segment)或称“分词”,即把它分成一个词的序列。如上例,则为“网络与 分布式 系统 实验室”(注意,不同的分词软件可能得出不同的结果,这里用的是北大计算语言所的在线分词软件)。然后需要删除那些没有查询意义或者几乎在每篇文档中都会出现的词(如“的”),在本例中即为“与”。最后形成一个用于参加匹配的查询词表,  $q = \{t_1, t_2, \dots, t_m\}$ ,在本例中就是  $q = \{\text{网络, 分布式, 系统, 实验室}\}$ 。前面讲过,倒排文件就是用词来作为索引的一个数据结构,显然,  $q$  中的词必须是包含在倒排文件词表中才有意义。有了这样的  $q$ ,它的每一个元素都对应倒排文件中的一个倒排表(文档编号的集合),记作  $L(t_i)$ ,它们的交集即为对应查询的结果文档集合,从而实现了查询和文档的匹配。上述过程的基本假设是:用户是希望网页包含所输入查询文字的。

## 2. 结果排序

上面,我们了解了得到和用户查询相关的文档集合的过程。这个集合的元素需要以一定的形式通过计算机显示屏呈现给用户。就目前的技术情况看,列表是最常见的形式(但人们也在探求新的形式,如 Vivisimo 引擎将结果页面以类别的形式呈现)。给定一个查询结果集合,  $R = \{r_1, r_2, \dots, r_n\}$ ,所谓列表,就是按照某种评价方式,确定出  $R$  中元素的一个顺序,让这些元素以这种顺序呈现出来。笼统地讲,  $r_i$  和  $q$  的相关性(relevance)是形成这种顺序的基本因素。但是,有效地定义相关性本身是很困难的,从原理上讲它不仅和查询词有关,而且还和用户的背景,以及用户的查询历史有关。不同需求的用户可能输入同一个查询,同一个用户在不同的时间输入的相同的查询可能是针对不同的信息需求。为了形成一个合适的顺序,在搜索引擎出现的早期人们采用了传统信息检索领域很成熟的基于词汇出现频度的方法。大致上讲就是一篇文档中包含的查询( $q$ )中的那些词越多,则该文档就应该排在越前面;再精细一些的考虑则是:若一个词在越多的文档中有出现,则该词用于区分文档相关性的作用就越小。这样一种思路不仅有一定直觉上的道理,而且在倒排文件数据结构上很容易实现。因为,当我们通过前述关键词的提取过程,形成一篇文档的关键词集合,  $p = \{t_1, t_2, \dots, t_n\}$  的时候,很容易同时得到每一个  $t_i$  在该文档中出现的次数,

即词频,而倒排文件中每个倒排表的长度则对应着一个词所涉及的文档的篇数,即文档频率。然而,由于网页编写的自发性、随意性较强,仅仅针对词的出现来决定文档的顺序,在 Web 上做信息检索表现出明显的缺点,需要有其他技术的补充。这方面最重要的成果就是前面提到过的 PageRank。通过在预处理阶段为每篇网页形成一个独立于查询词(也就和网页内容无关)的重要性指标,将它和查询过程中形成的相关性指标结合形成一个最终的排序,是目前搜索引擎给出查询结果排序的主要方法。

### 3. 文档摘要

搜索引擎给出的结果是一个有序的条目列表,每一个条目有三个基本的元素:标题、网址和摘要。其中的摘要需要从网页正文中生成。一般来讲,从一篇文字中生成一个恰当的摘要是自然语言理解领域的一个重要课题,人们已经做了多年的工作并取得了一些成果。但相关的技术用到网络搜索引擎来有两个基本困难。一是网页的写作通常不规范,文字比较随意,因此从语言理解的角度难以做好;二是复杂的语言理解算法耗时太多,不适应搜索引擎要高效处理海量网页信息的需求。我们做过统计,即使是分词这一项工作(文本理解的基础),在高档微机上每秒钟也只能完成 10 篇左右网页的处理。因此搜索引擎在生成摘要时要简便许多,基本上可以归纳为两种方式:一是静态方式,即独立于查询,按照某种规则,事先在预处理阶段从网页内容提取出一些文字,如截取网页正文的开头 512 字节(对应 256 个汉字),或者将每一个段落的第一个句子拼起来,等等。这样形成的摘要存放在查询子系统中,一旦相关文档被选中与查询项匹配,就读出返回给用户。显然,这种方式对查询子系统来说是最轻松的,不需要做另外的处理工作。但这种方式的一个最大的缺点是摘要和查询无关。一篇网页有可能是多个不同查询的结果,例如,当用户分别查询“北大计算机网络”和“北大分布式系统”,我们实验室的主页 <http://net.pku.edu.cn> 在两种情况下应该都作为结果返回。当用户输入某个查询,他一般是希望摘要中能够突出显示和查询直接对应的文字,希望摘要中出现和他关心的文字相关的句子。因此,我们有了“动态摘要”方式,即在响应查询的时候,根据查询词在文档中的位置,提取出周围的文字来,在显示时将查询词标亮。这是目前大多数搜索引擎采用的方式。为了保证查询的效率,需要在预处理阶段分词的时候记住每个关键词在文档中出现的位置。

除上述外,查询服务返回的内容还有一些细节的支持。例如,对应一个查询往往会有成千上万的结果,返回给用户的内容通常都是按页组织的,一般每页显示 10 个结果。统计表明(Wang et al. 2001),网络用户一般没有耐心一页页看下去,平均翻页数小于 2。这告诉我们将第一页的内容组织好非常重要。如果希望用户多用搜索引擎,就要让第一页的内容尽量有吸引力。

## 第五节 体系结构

在上述工作原理的基础上,作为一个网络应用软件,我们可以勾画出搜索引擎的体系结构,如图 2-3 所示,其中的大部分模块和前面的原理描述有直接的对应。这里需要特别讨论的是还没有专门提及的“控制器”模块。

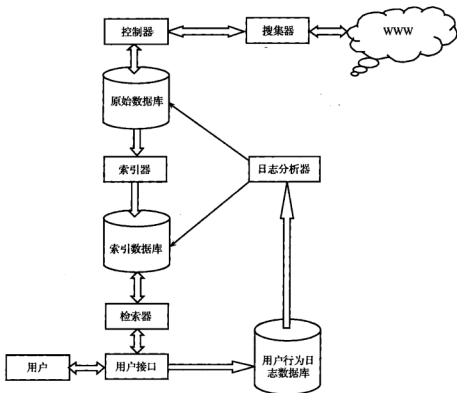


图 2-3 搜索引擎的体系结构

网页的搜集,如果只是为了做些简单的实验,不过上万篇网页的话,许多矛盾都不会出现,可以用最简单的工具(如 wget<sup>①</sup>)完成。但如果是为了向大规模搜索引擎稳定地提供网页数据,通常需要每天搜集上百万网页,而且是持续进行,情况则要复杂许多,核心是要综合解决效率、质量和“礼貌”的问题。这就是“控制器”的作用。

所谓效率,在这里就是如何利用尽量少的资源(计算机设备、网络带宽、时间)来完成预定的网页搜集量。在批量搜集的场合,我们通常考虑半个月左右能搜集到的网页,自然是越多越好。由于网页之间存在的独立性,利用许多台计算机同时来做这

<sup>①</sup> <http://www.gnu.org/software/wget/wget.html>

项工作是一个吸引人的想法,在第六章有专门的论述。这里需要指出三点:第一,即使是用一台计算机来搜集网页,也应该注意并发性的开发和利用。由于从网上抓取一篇网页通常需要秒量级的等待网络通信时间,同时启动多个抓取进程/线程,或者利用操作系统提供的异步通信机制,让多个网络通信时间重叠起来,让网络通信时间和存放网页的磁盘访问时间重叠起来是很有意义的。同时启动抓取进程的数量取决于硬件条件和搜集软件的设计,一般情况下进程数量可达上百个,做得好也可能达到上千个(即上千个进程也不会造成 CPU 成为瓶颈)。

在效率问题上应该指出的第二点是并不是设备越多越好。在用若干台计算机形成一个机群的安排下,它们共同分享出口网络带宽,随着设备量的增加,这个网络带宽(或者是周围的某个环境带宽)很快就成为瓶颈。经验表明实际上用不了超过 10 台计算机。人们曾经有过分布式搜集的想法,即让多台设备分布在网络上的不同位置,从而克服上述带宽瓶颈问题。理论上是可以的,但对于维护千万到亿量级的搜索引擎来说似乎还用不上,具体实现起来的麻烦会超过可能带来的好处(也许 Google 那样的针对多个国家用户的巨型搜索引擎需要用这种技术)。

影响搜集效率的第三点发生在网络的另一端,即服务器方,它可能来不及提供所需的网页。这除了有些 Web 服务器所处的网络条件比较差,或者有太多其他人访问外,搜索引擎太频繁对它们发出网页请求也是一个重要原因。落实到技术上,就是要有一个访问策略或者 URL 规划,不要让搜集器启动的抓取进程都集中在少数几个网站上。

将对搜集活动的关注过分集中在几个网站上,或者在一小段时间里从一个网站抓取太多的网页还可能引起其他的严重后果,即所谓“礼貌”问题。一般来讲,网站的管理人员都很愿意让自己的网页被搜索引擎索引,从而有可能得到更多的访问流量;但这只是问题的一方面。问题的另一方面是网站绝不希望由于搜索引擎的“密集”抓取活动阻碍了普通用户通过浏览器的访问,使那些用户得到这个网站访问起来很困难的印象,从而不再光顾。不加控制的网页抓取,给网站造成的现象有时候和制造拒绝服务(Denial of Service, DoS)攻击的黑客造成的现象一样。因此,管理良好的网站常常会有一个监视器运行,监视是否有来源于单个 IP 地址过分密集的访问。一旦出现这种情况,要么会通告该 IP 地址的拥有者注意行为,或者会干脆屏蔽来自它的访问,更有甚者,还可能在网上公布该 IP 地址作为黑名单。因此,适当地规划网页的抓取,限制单位时间内对一个网站抓取网页的数量(例如,每天不超过 2 万个,或者至少每隔 30 秒才对同一个网站发出下一个网页请求等),是大规模搜索引擎必须要认真对待的问题。总之,搜索引擎需要和网站“和睦相处”,它们是相互依存的。

所谓质量问题,指的是在有限的时间,搜集有限的网页,希望它们尽量是比较“重要”的网页,或者说不要漏掉那些很重要的网页。哪些网页是比较重要的?也是仁者见仁,智者见智的,不可能有一个统一认可的标准。如果让重要性和流行度等同起来,即越多人看过的网页越重要,至少在直觉上是有一定道理的。这样,我们可以考

虑一个网站从主页开始向下,按照链接的深度将网页组织成一层层,上层中的网页统计上会比下层的网页重要些。这样一种认识通过 PageRank 得到了加强,即较靠近主页的网页通常 PageRank 值较高。这样,首先得到尽量多的主页,然后从主页开始的先宽搜索就应该是一个较好的策略。闫宏飞论文(闫宏飞 2002)是支持这种观点的。

网页搜集过程中还有一个基本的问题是要保证每个网页不被重复抓取。由于一篇网页可能被多篇网页链接,在 spider 爬取过程中就可能多次得到该网页的 URL。于是如果不加检查和控制,网页就会被多次抓取。遇到循环链接的情况,还会使爬取器陷死。解决这个问题的有效方法是使用两个表,unvisited\_table 和 visited\_table。前者包含尚未访问的 URL,后者记录已访问的 URL。系统首先将要搜集的种子 URL 放入 unvisited\_table,然后 spider 从其中获取要搜集网页的 URL,搜集过的网页 URL 放入 visited\_table 中,新解析出的并且不在 visited\_table 中的 URL 加入 unvisited\_table。此方法简单明了,适合在单个节点上实现。但是当搜集子系统涉及多个节点的时候,如何避免各个节点之间的重复工作就复杂了,还要考虑网络的通信量、负载平衡,以及单个节点性能瓶颈等问题。这些问题的细节将在第六章讨论。

## 第六节 小 结

搜索引擎三段式是网页搜集、预处理和查询服务。在网页搜集阶段要考虑搜集来的网页时新性和大规模全网搜集网页的策略。在预处理阶段,需要考虑关键词提取、重复网页消除、链接分析和网页重要度计算。在查询服务阶段,涉及查询方式和匹配、结果排序和文档摘要问题。

本章最后给出了一个典型搜索引擎的体系结构图。

## 第三章 Web 信息的搜集

从第二章的学习中,我们知道搜索引擎一般采用三段式的工作流程,即:网页搜集、预处理和查询服务。为了便于读者对搜索引擎技术很快有一个全面实在的掌握,从本章开始的连续三章,我们讲解一个完整的搜索引擎 TSE(Tiny Search Engine)的实现,编程语言采用 C++,代码可以在(Tse 2004)下载。TSE 包括三段式工作流程,分别对应本章的 Web 信息的搜集,第四章搜集信息的预处理和第五章的信息查询服务。

作为三章内容的一个起始,在本章第一节中简单介绍互联网上的 HTTP 协议和 TSE 系统框架。然后主要内容集中在 TSE 的 Web 信息搜集部分。在后续的章节叙述中,文档也是指网页,我们不加以区分。

### 第一节 概 述

#### 一、超文本传输协议

超文本传输协议(Hypertext Transfer Protocol, HTTP)<sup>①</sup>是 Web 的基础协议。为了本章的完整,首先对 HTTP 进行简要的介绍,然后重点讲解如何实现 Web 信息的收集。

HTTP 是一个简单的协议。客户进程建立一条同服务器进程的 TCP 连接,然后发出请求并读取服务器进程的应答。服务器进程关闭连接表示本次响应结束。服务器进程返回的内容包含两个部分:一个“应答头”(response header),一个“应答体”(response body),后者通常是一个 HTML 文件,我们称之为“网页”。

在 Linux(或 Windows)环境下,有一个简单的方法可以让我们来感受一下 HTTP 协议的工作情况,即运行一个 Telnet 的客户程序与一个 HTTP 服务器程序通信。

下面是获取北京大学主页的例子(注意,下面显示的从服务器得到的内容不包括上述“应答头”)。

```
[webg@BigPc]$ telnet www.pku.cn 80 // 连接到服务器的 80 号端口
```

---

<sup>①</sup> 关于 HTTP 协议的详细介绍可以参看 RFC2068 Hypertext Transfer Protocol—HTTP/1.1 (RFCs 2004)。此外,在很多书中都有专门的章节进行介绍,如《TCP-IP 详解卷三:TCP 事务协议,HTTP,NNTP 和 UNIX 域协议》的第 13 章 HTTP:超文本传送协议;《UNIX 技术大全—Internet 卷》的第 21 章超文本传输协议简介。

```
Trying 162.105.129.12...           // 由 Telnet 客户输出
Connected to rock.pku.cn (162.105.129.12). // 由 Telnet 客户输出
Escape character is '^]'.           // 由 Telnet 客户输出
GET /                               // 我们只输入了这一行
<html>                             // Web 服务器输出的第一行
<head>
<title>北京大学</title>
.....                             // 这里省略了很多行输出
</body>
</html>
Connection closed by foreign host. // 由 Telnet 客户输出
```

我们只输入了 GET /, 服务器却返回了很多字节。这样, 从该 Web 服务器的根目录下取得了它的主页。Telnet 的客户进程输出的最后一行信息表示服务器进程在输出最后一行后关闭了 TCP 连接。

一个完整的 HTML 文档以<HTML>开始, 以</HTML>结束。大部分的 HTML 命令都像这样成对出现。HTML 文档含有以<HEAD>开始、以</HEAD>结束的首部和以<BODY>开始、以</BODY>结束的主体部分。标题通常由客户程序显示在窗口的顶部。关于 HTML 规范的详细介绍可以参看 (W3C 1999)。

在接下去的几节中, 将通过一个小的搜索引擎系统 TSE (运行在 Red Hat Linux 8.0 以上的系统中) (Tse 2004) 的循序渐进实现, 一边讲原理技术, 一边讲代码, 描述 Web 信息搜集的过程, 本处的 Web 信息搜集主要指网页信息。

网页搜集子系统, 就是第一章第二节和第二章第五节中讲到的 spider, 可以用 C/C++、Perl、Java、Python 等语言来编写, 可以运行在 Intel、Sparc、Mac 等平台上的 Unix 或 Window 系统下。网页“爬取器”(gatherer), 指网页搜集子系统中根据 URL 完成一篇网页抓取的进程或者线程, 通常一个 spider 会同时启动多个 gatherer 并行工作。Spider 设计是否合理将直接影响它访问 Web 的效率, 影响搜集数据的质量。另外, 在设计 spider 时还必须考虑它对网络和被访问站点的影响, 因为 spider 一般都运行在速度快、带宽高的主机上, 如果它快速访问一个速度比较慢的目标站点, 就有可能导致该站点出现拥塞甚至宕机。Spider 还应遵守一些协议 (如 robot 限制协议 (Wong 1997)), 尊重被访问站点管理员确定的内容保护策略。

## 二、一个小型搜索引擎系统

TSE 是一个适合教学用的搜索引擎, 设计的目标之一是让它足够小, 便于任何一个对搜索引擎感兴趣的人都可以利用自己有限的硬件资源 (如自己的台式机) 搭建; 让它尽量简单, 让具有一般程序设计基础的爱好者可以全部理解; 让它的功能相对完整, 能够反映一个大规模搜索引擎的主要成分。



首先从 TSE 的外部表现形式来看它所能完成的工作,然后给出 TSE 系统结构图。搜索引擎是通过浏览器界面展现给用户的,图 3-1 是 TSE 的用户界面。



图 3-1 TSE 搜索引擎界面

在查询输入框输入查询短语并回车(或者点击“搜索”按钮),即可得到相关资料。查询时关键词之间不需要使用“and”,因为 TSE 会在关键词之间自动添加“and”。TSE 提供符合您查询条件的全部网页。

例如,想查北大校庆,只需在图 3-1 的搜索框中输入“北大校庆”,然后回车。图 3-2 是 TSE 的查询返回结果。为方便解释起见,我们用 A、B、C 标识其中的几个部分。

1) A 表示统计栏,包括用户输入的查询词,有关查询结果和搜索时间(一般搜索响应时间不超过 1 秒钟)的统计数字;

2) B 表示一条查询结果,包括该网页网址、网页摘要(在摘要信息中,您的原始查询字词,都用红色字体表示,以便阅读)。

3) C 表示网页快照。通过链接访问网页失效时,可以访问 TSE 的缓存网页;或者网络拥塞的时候,可以通过访问缓存网页避免直接访问该网页。

点击图 3-2 中的“网页快照”链接,得到图 3-3 所示的 TSE 网页快照页面。图中最上部分标明此网页来自 TSE 的网页快照。用户输入的查询短语如果被系统分成多个关键词,用不同颜色表示,并增加链接便于点击相应关键词直接到达正文中该关键词出现的位置。正文部分取自网页原文的缓存,其中包含用户查询关键词的文字加亮显示。

图 3-1、图 3-2、图 3-3 展示了 TSE 的查询服务功能,为了完成上述功能,需要网页搜集和预处理两个部分的支持。图 3-4 所示为 TSE 系统结构,对应于搜索引擎三

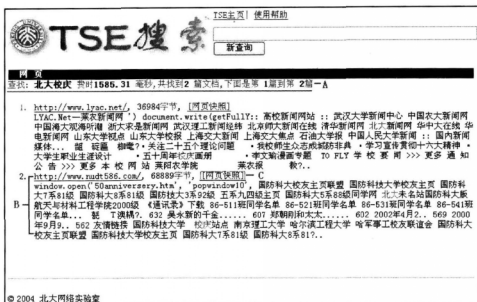


图 3-2 TSE 查询结果页面

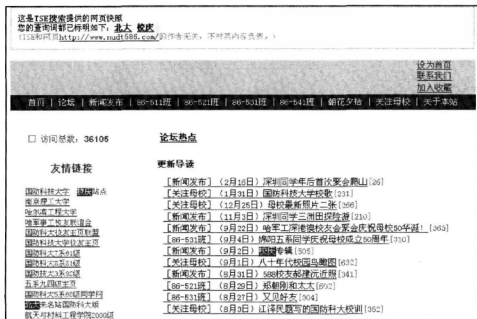


图 3-3 TSE 网页快照页面

段式工作流程,图中左侧的 A 表示搜集部分,中间的 B 表示整理(即预处理)部分,右侧的 C 表示服务部分。其中圆柱形表示数据产品,按照统一并且简单易懂的格式存储,除本系统使用外,可以提供给其他科研机构使用;椭圆形表示系统流程中的内部数据,由于与系统中使用的数据结构结合紧密,不适合作为数据产品提供给其他研究机构;矩形表示系统流程的程序部分(过程),是数据产品与内部数据之间的桥梁。系统起始于 A 搜集,结束于 C 服务,整个流程可以重复进行,从而达到系统的更新。图 3-4 中的各个数据产品,内部数据和过程在后续章节相应部分细致讲解。在 TSE 中不包括 PageRank 的计算和日志挖掘,这两个过程主要是对查询结果的排序产生作用,在实际应用的搜索引擎中是必不可少的,但是对于讲解搜索引擎的工作过程不是必需的。

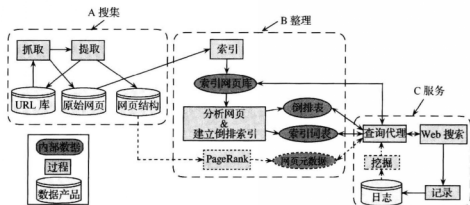


图 3-4 TSE 系统结构

本章后续内容讲解搜集部分(对应图 3-4 中的 A);第四章讲解预处理部分(对应图 3-4 中的 B);第五章讲解服务部分(对应图 3-4 中的 C)。

## 第二节 网页搜集

在 TSE 中网页搜集对应图 3-4 的左侧部分,我们将它细化,如图 3-5 所示。网页搜集的过程是从 URL 库(初始时包含用户指定的起始种子 URL 集合,可以是一个或多个)获得输入,解析 URL 中标明的 Web 服务器地址、建立连接、发送请求和接收数据,将获得的网页数据存储在原始网页库,并从中提取出链接信息放入网页结构库,同时将待抓取的 URL 放入 URL 库,保证整个过程的递归进行,直到 URL 库为空。

搜索引擎为了提供检索服务,需要保存网页原文。网页搜集子系统不但要能够获取以 .html、.htm、.txt 结尾的 URL 对应的网页(在本章后面的小节对于搜集信息类型会有更详细的阐述),还应该能够获取不是以 .html 结尾的 URL,如 .pdf、

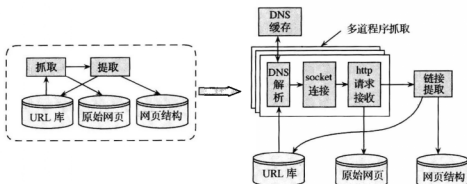


图 3-5 Web 信息的搜集

.doc、因为.pdf、.doc 等文件可以通过转换程序生成为.html 或者.txt 文件,同样为搜索引擎提供检索服务。作为搜索引擎的起始流程,搜集的网页要按照一定的格式存储,便于后续组织和提供服务。

传统应用的实现在系统结构定下来后,重点是确定其中的数据结构。而面向对象方法具有更好的特性,我们是采用 C++ 面向对象的方法结合源代码讲述。针对系统实现的描述,依据重要名词定义为类的原则,重点讲解 URL 类和 Page 类。本章后续部分是把 TSE 中的搜集部分分解开来讲解,包括:定义 URL 类,定义 Page 类,与服务器建立连接,构造请求消息体并发送给服务器,获取服务器返回的网页元信息(或者称为网页头信息)和获取网页信息(或者称为网页体信息)。

## 一、定义 URL 类和 Page 类

### 1. 定义 URL 类

TSE 程序首先必须要做的一件事情是根据一个给定的 URL 组成消息体,发送给该 URL 指向的服务器。为此,定义 URL 类。关于 URL 的详细介绍可以参看 RFC2396 Uniform Resource Identifiers (URI): Generic Syntax(RFCs 2004)。

下面是 URL 类的定义,对应文件 Url.h。

```
enum URL_scheme{
    SCHEME_HTTP,
    SCHEME_FTP,
    SCHEME_INVALID
};

class CUrl
{
```

```

public:
    string m_sUrl;                // URL 字符串
    enum URL_scheme m_eScheme;   // 协议名
    string m_sHost;               // 主机名
    int m_nPort;                  // 端口号
    string m_sPath;               // 请求资源

public:
    CUrl();
    ~CUrl();
    bool ParseUrl( string strUrl );

private:
    void ParseScheme ( const char * URL );
};

```

URL 可以是 HTTP、FTP 等协议开始的字符串, TSE 主要是针对 HTTP 协议, 为了不失一般性, 在 URL\_scheme 中定义了 SCHEME\_HTTP, SCHEME\_FTP, SCHEME\_INVALID, 分别对应 HTTP 协议, FTP 协议和其他协议。一个 URL 由 6 个部分组成:

<scheme>://<net\_loc>/<path>,<params>? <query>#<fragment>

除了 scheme 部分, 其他部分可以不在 URL 中同时出现。

Scheme 表示协议名称, 对应于 URL 类中的 m\_eScheme。

Net\_loc 表示网络位置, 包括主机名和端口号, 对应于 URL 类中的 m\_sHost 和 m\_nPort。

下面四个部分对应于 URL 类中的 m\_sPath。

Path	表示 URL 路径。
Params	表示对象参数。
Query	表示查询信息, 也经常记为 request。
Fragment	表示片断标识。

为了程序的简化, URL 类的实现主要是解析出 net\_loc 部分, 用于组成消息体, 发送给服务器。

其中 void CUrl::ParseUrlEx(const char \* URL, char \* protocol, int lprotocol, char \* host, int lhost, char \* request, int lrequest, int \* port) 执行具体的字符串匹配找出协议名, 主机名, 请求信息和端口号, 找到后赋给 URL 类的成员变量保存。在 URL 类中还有些是 TSE 抓取过程中的细节函数, 如 char \* CUrl::GetIpByHost(const char \* host), CUrl::IsValidHost(const char \* host), bool CUrl::IsVisitedUrl(const char \* URL) 等, 此处就不一一介绍了。读者可以通过阅

读 TSE 的代码获得这部分的具体实现。

## 2. 定义 Page 类

有了 URL,搜集系统就可以按照 URL 标识抓取其所对应的网页,网页信息保存在 Page 类中。

下面是 Page 类的定义,对应文件 Page.h。

```
class CPage
{
public:
    string m_sUrl;

    // 网页头信息
    string m_sHeader;
    int m_nLenHeader;

    int m_nStatusCode;
    int m_nContentLength;
    string m_sLocation;
    bool m_bConnectionState; // 如果连接关闭,是 false,否则为 true
    string m_sContentEncoding;
    string m_sContentType;
    string m_sCharset;
    string m_sTransferEncoding;

    // 网页体信息
    string m_sContent;
    int m_nLenContent;
    string m_sContentNoTags;

    // link, in a lash-up state
    string m_sContentLinkInfo;

    // 为搜索引擎准备的链接, in a lash-up state
    string m_sLinkInfo4SE;
    int m_nLenLinkInfo4SE;

    // 为历史存档准备的链接, in a lash-up state
    string m_sLinkInfo4History;
    int m_nLenLinkInfo4History;
```

```
//为搜索准备的链接, in a good state
RefLink4SE m_RefLink4SE[MAX_URL_REFERENCES];
int m_nRefLink4SEnum;

//为历史存档准备的链接, in a good state
RefLink4History m_RefLink4History[MAX_URL_REFERENCES/2];
int m_nRefLink4HistoryNum;

map<string, string> m_mapLink4SE;
vector<string> m_vecLink4History;

enum page_type m_eType; // 网页类型

public:
    CPage();
    CPage::CPage(string strUrl, string strLocation, char * header,
        char * body, int nLenBody);
    ~CPage();

    void ParseHeaderInfo(string header); // 解析网页头信息
    bool ParseHyperLinks();           // 从网页体中解析链接信息

    bool NormalizeUrl(string& strUrl);
    bool IsFilterLink(string plink);

private:
    // 解析网页头信息
    void GetStatusCode(string header);
    void GetContentLength(string header);
    void GetConnectionState(string header);
    void GetLocation(string header);
    void GetCharset(string header);
    void GetContentEncoding(string header);
    void GetContentType(string header);
    void GetTransferEncoding(string header);

    // 从网页体中解析链接信息
    bool GetContentLinkInfo();
    bool GetLinkInfo4SE();
```

```
bool GetLinkInfo4History();  
bool FindRefLink4SE();  
bool FindRefLink4History();  
};
```

Page 类的具体实现请参看 Page.cpp 文件。一个网页是以 URL 作为标识的,所以 Page 类的第一个成员变量是 m\_sUrl。Page 类主要完成两个任务:解析网页头信息和提取链接信息。

解析网页头信息包括获得状态码 m\_nStatusCode,网页体长度 m\_nContentLength(内容字节数),转向信息 m\_sLocation,连接状态(如果没有关闭,下次请求同一个网站可以重新利用已经建立好的 Socket,节约资源),网页体编码 m\_sContentEncoding(如果是 gzip 编码,要解压缩,然后提取链接信息。现在门户网站的首页有增大趋势,为了加快传输速度,通常采用 gzip 编码压缩后传输),网页类型 m\_sContentType,网页体字符集 m\_sCharset 和传输编码方式 m\_sTransferEncoding。

提取链接信息,是从获得的网页体中,根据 HTML 的规定,提取出链接信息和相应的链接描述信息,形成网页结构库,扩充 URL 库。在 TSE 中对于网页内容的链接提取区分了为搜索引擎提取和为历史网页存档提取两种。因为对于通常意义下的搜索引擎而言,图片链接、网页格式链接是没有用处的,如果不区分开,会增加程序运行的负担,增加存储空间。而且区分开后,可以单独保存下来,便于以后单独搜集这些信息。

## 二、与服务器建立连接

已经从 URL 中获得了服务器的主机名,要能够从服务器上获取网页内容,还需要客户端进程与服务端进程建立连接。UDP 和 TCP 的通信采用 Socket 方法实现,Socket 为进程间通信提供了端点。通信由消息组成,消息是在一个进程的 Socket 与另一个进程的 Socket 之间传送的。如图 3-6 所示,一个进程要能够接收消息,它的 Socket 必须绑定到一个本地端口和本地地址上。发送到指定 Internet 地址和端口上的消息,只能被绑定到该地址和端口的 Socket 所属进程接收。

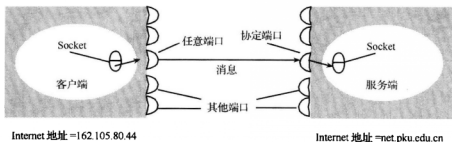


图 3-6 Sockets 和端口



连接的建立过程是异步的,如图 3-7 所示,一方在监听建立连接的请求,一方将发起建立连接的请求。连接一旦被接受,操作系统(如 UNIX)自动创建新的 Socket 使之与客户端连接成通信的通道,这样服务端就可以在原来的 Socket 上继续监听其他客户的请求了。连接建立后,双方进程可以通过建立好的连接进行读写操作。

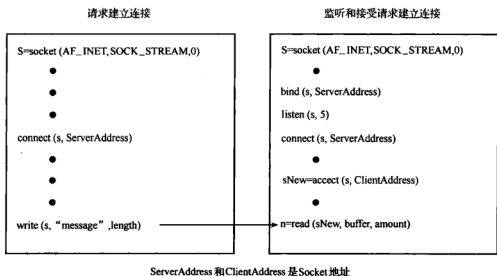


图 3-7 通过 Socket 建立连接

与服务器建立连接的函数 CreateSocket 在 Http.h 中定义,在 Http.cpp 中实现。与服务器建立连接使用了非阻塞连接,超过定时,就放弃。具体代码可以参看 Http.h 中定义的 int nonb\_connect(int, struct sockaddr \*, int) 函数。

考虑到此函数是为 Web 信息的搜集服务设计的,特别增加 DNS 缓存和 IP 范围控制功能。

增加 DNS 缓存的原因:

1) URL 数以亿计,而主机数以百万计。如果没有 DNS 缓存,搜集子系统每次搜集新的 URL,即使刚在上一次的任务中解析过该主机名,也要重新进行域名解析(除了用 IP 地址直接表示的主机,如 162.105.80.44)。为了避免频繁的查询 DNS 服务器,造成类似于拒绝服务攻击的副作用,要建立 DNS 缓存。

2) 针对小规模网页搜集(如百万量级),DNS 缓存只需要建立在内存中就够了,一是因为内存占用不多,二是因为域名解析有时效性,即使把这些解析出的信息保存到外存上,过几天也失去了意义。如果是为亿量级服务的 Web 信息搜集模块,要考虑单独建立为搜集子系统服务的 DNS 模块,这样能够既加快网页信息的获取,又进一步降低对 DNS 服务器的压力。

增加 IP 范围控制的原因:

1) 有些站点不希望搜集程序搜走自己的资源。如很多在大学范围内的论文检

索系统,通常只是对学校 IP 范围内开放的。如果搜集程序处于允许检索 IP 范围内,搜集了这些资源,然后提供服务,就有可能使非授权用户看到了论文信息。

2) 针对特定信息的搜索,如校园网搜索、新闻网站搜索。

3) 网络资费方式也会对搜集策略产生影响。有的网络基础设施提供商可能会规定在对不同 IP 范围信息的访问采用不同的资费政策,因此从运行成本考虑,也可能需要对 IP 范围进行控制。

### 三、发送请求和接收数据

#### 1. 构造请求消息体并发送给服务端

搜集端与服务端建立好连接后,前者按照 HTTP 的要求构造消息体发送给服务端。这段实现代码包含在函数 `int CHttp::Fetch( string strUrl, char ** fileBuf, char ** fileHeadBuf, char ** location, int * nPSock )` 中。

`HttpFetch` 函数中部分代码参考了 `http://fetch.sourceforge.net` 中的 `int http_fetch(const char * URL_tmp, char * * fileBuf)` 函数,但是它不能够返回请求 URL 的网页头信息,不能确定网页是否已经重定向,所以我们有针对性地做了改动。另外由于该函数是用 C 写成的,为了保持原代码的完整,对于 URL 的解析需要单独做一次,而不是采用前面已经给出的 URL 类来实现(URL 类在 TSE 的其他部分会经常用到)。

代码实现在文件 `Http.cpp` 中。在 TSE 中消息体的组装中采用 HTTP 1.1 协议,抓取程序不主动关闭 Socket 通道,这样当 Web 服务器支持持续连接,后续该网站的网页请求就可以利用已经建立的 Socket 通道进行,可以节省时间和网络带宽。如果服务器不支持持续连接,使用已经建立的 Socket 通道会报错,此时需要重新建立连接。

#### 2. 获取网页头信息和体信息

服务端接受搜集端发送的请求消息后,先返回一个 HTTP 头信息(为方便起见,后面我们称“网页头信息”),其中包含文件类型、大小、最后修改时间等内容;接着是一个“`\r\n`”,表现为一个空行;然后返回 HTTP 体信息,其中包含网页的全文内容。获取上述头信息和体信息的实现代码同样包含在函数 `int CHttp::Fetch(string strUrl, char ** fileBuf, char ** fileHeadBuf, char ** location, int * nPSock)` 中,在文件 `Http.cpp` 中。

网页头信息获取后,进行解析,根据返回码,判断 Web 服务器是否针对该请求转向,如果转向,应该重新组装消息体发送请求;根据传输类型,网页体的大小,申请内存空间准备接收,如果超出预定接收大小,放弃该网页;根据网页类型,判断是否获取该网页。如果满足获取条件,继续进行网页体信息的获取。

注意网页头信息中给定的网页体大小有可能错误,所以读取网页体信息应该是在一个循环体中,直到不能读到新的字节为止。因此根据需求增减内存空间,是格外需要注意的地方。在读取网页信息时,存在服务器长时间不响应的情况,为了加快搜集效率,要设定超时机制,超时后放弃该网页。接收的数据如果超出预定接收大小,放弃该网页。

#### 四、网页信息存储的天网格式

将获取的网页信息保存在磁盘中,需要按照规定的格式保存,便于后续处理和提供服务。在 TSE 中,采用天网格式存储获得的网页。下面介绍天网格式存储方案。注意这种方案只是顺序保存网页信息,没有索引文件。在第四章中我们会讲到,如何组织这些保存下来的原始网页信息,建立索引文件,支持单个网页信息的检索。

原始网页信息的存储格式应当设计为适合长期保存并易于处理,可以作为终端产品提供给用户使用。考虑到终端产品使用的便利性,要求原始网页库的存储格式具备简单性的特点。

由于存储介质都是有寿命的,所以应当考虑当存储介质损坏时数据的可恢复性。例如,磁盘的某个扇区损坏,导致部分数据不能读出,如果剩下的数据仍然可以使用,就能将损失降到最少。对于海量数据来说,在存储和传输的过程中,由于硬件和软件问题导致数据错误是不可避免的。因此,原始网页的存储格式还应当具备隔离错误的特点。

##### 1. 天网存储格式定义

根据以上考虑,天网存储格式定义如下:

1) 一个原始网页库(RAW\_DB)由若干记录组成,每个记录(RECORD)包含一个网页的原始数据,记录的存放是顺序追加的,记录之间没有分隔符;

2) 一个记录由头部(HEAD)、数据(DATA)和空行(BLANK\_LINE)组成,顺序是头部+空行+数据+空行;

3) 一个头部由若干属性组成,每个属性(PROPERTY)是一个非空的行,头部内部不允许出现空行;

4) 一个属性包含属性名(NAME)和属性值(VALUE),并由冒号“:”隔开,顺序是属性名+冒号+属性值;

5) 头部的第一个属性必须是版本属性,属性名为 version,如 version: 1.0,该属性表明记录的版本号;

6) 头部的最后一个属性必须是数据长度属性,属性名为 length,如 length: 1800,该属性值必须是数据(DATA)的长度(字节数),不包括空行的长度;

7) 为简化起见,属性名必须是小写的字符串。

注:一个空行(BLANK\_LINE)仅由一个换行符(line feed,LF,即 C 语言中的“\

n”)组成,在 UNIX 系统的显示中表现为一个空行,所以称为空行。Microsoft Windows 系统和 UNIX 系统在空行机制上有所区别:在 Windows 系统下,纯文本显示中的一个空行由一个回车符(carriage return,CR)和一个换行符组成(即 C 语言中的“\r\n”);而在 UNIX 系统中一个空行仅由一个换行符组成。

## 2. 当前存储格式版本描述

存储格式允许有多个版本,以满足将来进行扩展的需要。

当前存储格式的版本属性为 1.0。一个记录的存储格式如下(// 后为注释):

```
version: 1.0                // 版本号
url: http://www.pku.edu.cn/ // URL
origin: http://www.somewhere.cn/ // 原来的 URL
date: Tue, 15 Apr 2003 08:13:06 GMT // 抓取时间
ip: 162.105.129.12         // IP 地址
unzip-length: 30233        // 如果数据经过压缩,则需有此属性
length: 18133              // 数据长度
                             // 空行
XXXXXXXXX                  // 以下为数据
XXXXXXXXX
.....
XXXXXXXXX                  // 数据结束
                             // 最后再插入一个空行
```

各属性说明:

version 属性为版本号,以下说明适用版本号为 1.0 的情况。

url 指该网页的 URL,如果因为网页头信息中包含 Location 字段而产生网页转向时,该 URL 为最后实际抓取的 URL 地址。该属性是必需的。

origin 指该网页的原始 URL。该属性仅在 HTTP HEAD 中包含 Location 字段而产生网页转向时存在,指向最原始的 URL。

date 属性为该网页的保存时间,保存格式为 RFC822 所制定的格式。该属性是必需的。

ip 属性为该网页所在服务器的 IP 地址。

unzip-length 属性仅在数据经过压缩时存在,记录数据未压缩时的原始长度。

length 属性记录数据长度。

若存在其他未加说明的属性,应用程序可以简单地忽略。

关于数据是否压缩的问题:天网格式并不指定数据是否必须经过压缩。但是压缩的数据必须包含 unzip-length 属性而未压缩的数据不能包含该属性。该属性同时也是解压缩所必需的。如果数据经过压缩,还应附带说明压缩算法,必要时附带压缩

函数库及源代码。

### 3. 数据的可恢复性分析

假设由于数据遭到破坏,只得到其中一个残存的片段。则可按以下步骤找出该残存片段中所有完整的记录:

1) 特定字符串“version:”,除非没有一个完整的记录,否则该字符串肯定能找到。记录该字符串的位置 POS。

2) 找到该字符串后,判断其后的数据是否满足存储格式 2)、3)、4)、6)、7) 条件。如果任何一个条件不满足,返回步骤 1),从记录的位置 POS 开始继续查找下一个特定字符串“version”。

3) 当满足步骤 2) 条件时,假定这是一个正确的记录,则下一个记录也必定是一个正确的记录。检查该记录满足天网存储格式 2)、3)、4)、5)、6)、7) 条件,如果任何一个条件不满足,说明原先的假定错误,返回步骤 1),从记录的位置 POS 开始,继续查找下一个特定字符串“version”。如果条件都满足,则继续检查下一个记录是否正确。

4) 如果连续 3 个记录都是正确的,则认为步骤 1) 所找到的“version”是一个正确的记录的开始,可以依此提取出全部正确的原始网页。

由于原始网页是随机的,而存储格式是严格的,因此经过上述方法得到的记录为错误记录的可能性极小,是完全可以接受的。

### 4. 其他问题

在实际应用天网存储格式时,应该注意下面两个方面。

1) 文件打开模式。用标准 IO 库打开文件时有两种模式:文本(text)模式和二进制(binary)模式。在 UNIX 下这两种模式并没有区别,但在 Windows 下如果用文本模式打开,“\r\n”会被当成一个字符,而天网格式中的“length”域表示的是实际字节数,这就可能引起错误。因此,在用标准 IO 读取文件时,为了兼容性最好用二进制模式打开,例如 FILE \*fp=fopen(“filename”,“rb”)。

2) FTP 传输。FTP 传输也有两种传输模式:文本(text)模式和二进制(binary)模式。传输原始网页库文件时,应以二进制模式进行传输。如果以文本模式传输,可能会出现“\r\n”被替换为“\n”或“\n”被替换为“\r\n”的现象,导致数据错误。

## 第三节 多道搜集程序并行工作

搜集端程序相当于客户端,在 HTTP1.0 中,即使客户端希望在同一次会话中从同一服务器传输更多的 HTML 页面,该 TCP 连接也会被终止,每一个新的请求需要建立另一个 TCP 连接,这造成了 HTTP 服务器的负担。在 HTTP 1.1 版中,提供

对持续 TCP 连接的支持,可以参看 RFC2068 Hypertext Transfer Protocol—HTTP/1.1(RFCs 2004)。这样改变一下可节省 Web 服务器资源,而且可以节省网络可使用的带宽。此外,由于避免了每次请求都重新建立连接的开支,使用一个持续的连接比 HTTP/1.0 的实现具有更高的操作效率。在 TSE 中使用的是 HTTP/1.1 的请求方式,注意不是简单的更换 HTTP/1.0 这个字串为 HTTP/1.1,而是需要保留上次已经建立的连接,如果该连接没有失效,则本次继续使用。

通常情况下局域网的延迟在 1~10ms,带宽为 10~1000Mbit/s,Internet 的延迟在 100~500ms,带宽为 0.010~2 Mbit/s。所以针对搜索引擎应用的搜集程序通常是在同一个 LAN 内的多台机器,每个机器多个进程并发的工作。这样一方面可以利用 LAN 的高带宽、低延时,各节点充分交流数据,另一方面采用多进程并发方式降低 WAN 高延迟的副作用。因为 LAN 的利用率几乎接近 1,而 WAN 因为有路由和拥塞控制等额外要求利用率不高,所以 LAN 与 WAN 的连接是至关重要的。因此需要同时启动多个搜集程序并发的创建多个 TCP 连接,并发的下载网页。这种方式加快了 Web 信息的搜集,但是要避免多个搜集程序重复的收集网页(在本章第四节中具体介绍解决方法),还要避免由于同一时间内与同一服务器连接过多而给服务器端造成的严重性能问题。

究竟应该有多少个节点并行搜集网页,每个节点启动多少个搜集程序?下面逐步分析得到。先给出理论值,然后给出经验值,最后给出单节点的搜集效率。

### 一、多线程并发工作

#### (1) 计算理论值

天网 2002 年 1 月统计纯文本网页平均大小为 13KB(Yan et al. 2002)(下文中不特殊说明的网页均指纯文本网页)。

在连接速率为 100Mbit/s 快速以太网络上(以太网的 MTU(Maximum Transfer Unix)是 1500B),假设线路的最大利用率是 100%,则最多允许每秒传输  $(1.0 \times 10^8 \text{ bit/s}) / (1500 \text{ B} \times 8 \text{ bit/B}) \approx 8333$  个数据帧,也即每秒传输 8333 个网页。

如果 LAN 与 WAN 的连接为 100Mbit/s, WAN 带宽利用率低于 50%,则每秒传输的网页数目平均不到 4000 个。

如果搜集系统由  $n$  个节点组成,单个节点启动的搜集程序数目应该低于  $4000/n$ 。

#### (2) 计算经验值

在实际的分布式并行工作的搜集节点中,还要考虑 CPU 和磁盘的使用率问题,通常 CPU 使用率不应该超过 50%,磁盘的使用率不应该超过 80%,否则机器会响应很慢,影响程序的正常运行。在天网的实际系统中局域网是 100Mbit/s 的以太网,假设局域网与 Internet 的连接为 100Mbit/s(这个数据目前不能得到,是我们的估计),启动的搜集程序数目少于 1000 个。这个数目的搜集程序对于上亿级的天网索引

擎(天网 2004)是足够的。

### (3)单节点的搜集效率

以太网数据帧的物理特性是其长度必须在 46 ~ 1500 字节之间(Stevens 1996)说明在一个网络往返时间 RTT 为 200ms 的广域网中,服务器处理时间 SPT 为 100ms,那么 TCP 上的事务时间就大约 500ms( $2 \times \text{RTT} + \text{SPT}$ )。网页的发送是分成一系列帧进行的,则发送 1 个网页的最少时间是  $(13\text{KB}/1500\text{B}) \times 500\text{ms} \approx 4\text{s}$ 。如果系统处于满负荷运转,单个节点启动 100 个搜集程序,则每个节点每天应该能够搜集  $(24 \times 60 \times 60\text{s}/4\text{s}) \times 100 \text{ 页} = 2\,160\,000 \text{ 个网页}$ 。考虑到搜集程序实际运行中可能存在超时、搜集的网页失效等原因,每个节点的搜集效率小于 2 160 000 个网页/天。

### 启动多线程并发工作

在 TSE 中启动多个 gatherer 是用多线程方式实现的。

代码如下,在文件 Crawl.cpp 中。

```
// 创建多个搜集线程。
Pthread_t tids = (pthread_t*)malloc(NUM_WORKERS * sizeof(pthread_t));
if( tids == NULL)
    cerr << "malloc error" << endl;

for(unsigned int i=0; i< NUM_WORKERS; i++){
    if (pthread_create(&tids[i], NULL, start, this))
        cerr << "create threads error" << endl;
}
```

对应 start 的工作代码实现功能是多个搜集线程并发的从待抓取的 URL 队列中取出任务,然后去抓取 URL 对应的网页。请参考 start 过程。

## 二、控制对一个站点并发搜集线程的数目

提供 Web 服务的机器,能够处理的未完成的 TCP 连接数有一个上限,未完成的 TCP 连接请求放在一个预备队列,这个队列有一个预先定义的数量限制,它规定在任何时刻能够处理的请求数目。当一个服务器的预备队列达到它定义的限制时,任何新的连接请求都会被忽略,直到队列可用为止。

多道搜集程序并行的工作,如果没有控制,势必造成对于搜集站点的类似于 DOS 攻击的副作用,也就是塞满了 Web 服务器的预备队列,导致后续请求被忽略。

因此在 TSE 的 Crawl.cpp 中增加一段控制代码。

其中 NUM\_WORKERS\_ON\_A\_SITE,定义在文件 Tse.h 中,为 const unsigned int NUM\_WORKERS = 2,表示一个站点同一时刻最多有 2 个搜集程序在搜集它的网页。代码的实现是根据待放入 URL 的主机部分,判断待访问队列中是否已经有足够数目的包含该主机的 URL,如果是则等待,直到有空缺才插入待访问队列。空

缺的产生是因为搜集程序下载了该 URL,并从待访问队列删除了该 URL。

## 第四节 如何避免网页的重复搜集

重复搜集,是指物理上存在的一个网页,在没有更新的前提下,被搜集程序重复访问。造成重复搜集的原因,一方面是搜集程序没有清楚记录已经访问过的 URL,另一方面是由于域名与 IP 多重对应关系造成的。下面分情况介绍解决方法。

### 一、记录未访问、已访问 URL 和网页内容摘要信息

搜集程序从一个事先制定好的 URL 列表出发(可以理解为初始的未访问 URL 列表),这个列表中的 URL 通常是从以往访问记录中提取出来的,特别是一些热门站点和“What's New”网页,此外,很多搜索引擎还接受用户提交的 URL。搜集程序访问了一个网页后,会对它进行分析,提取出新的 URL,将之加入到待访问列表中,如此递归地访问 Web。

完成搜集网页的工作,即使是只有一个搜集程序,也需要解决如何避免重复搜集网页的问题。因此定义两个表,“未访问表”和“已访问表”。“未访问表”中存储准备取入待访问队列的 URL,“已访问表”中存储已经请求过网页的 URL。这样当搜集程序因为访问新的网页解析出新的 URL 后,根据“未访问表”,“已访问表”,就可以判断哪些工作已经完成,从而避免重复搜集。

在 TSE 中,除了存储上述“已访问表”和“未访问表”的摘要信息,还存储了已经获取网页内容的摘要信息。存储网页内容的摘要信息的原因是 Web 上存在大量的复制网页,它们的 URL 不同,内容完全一样。

在 TSE 中,对访问过的 URL,未访问过的 URL 和获得的网页内容分别作 MD5 摘要(算法可以参看 RFC1321)(RFCs 2004),获得其唯一标识,建立三个集合。新解析出的 URL,首先根据已经访问过的 URL 的 MD5 集合判断是否已经抓取过,如果没有则放入未访问 URL 库,否则放弃;查找的时候可以做到  $O(1)$  的时间复杂度。

### 二、域名与 IP 的对应问题

记录未访问和已访问 URL 信息,可以保证搜集的网页中所有的 URL 都不同。但是域名与 IP 的对应存在复杂的关系,导致即使 URL 不同,也可能指向的物理网页是相同的,这就导致重复搜集。为了解决这个问题,需要分析清楚域名与 IP 的对应关系。

域名与 IP 的对应关系存在四种情况:一对一、一对多、多对一、多对多。一对一不会造成重复搜集,后三种情况都有可能造成重复搜集。

后三种情况出现的原因:

1) 可能是虚拟主机。例如, www.pku.edu.cn、www.gh.pku.edu.cn、www.



acgs.pku.edu.cn 和 caspu.pku.edu.cn 都映射到同一个 IP 162.105.129.12 的情况,但这是采用虚拟主机技术完成的,由于 4 个站点的内容互不重复,认为是 4 个 Web 服务器。

2) 可能是 DNS 轮转。多域名的情况在商业站点中较常见,因为商业站点的单位时间访问量大,需要复制服务器内容,通过 DNS 轮转达到负载均衡,满足用户需求。

例如:

一个域名对应多个 IP。

```
[webg@BigPc]$ ping ent.sina.com.cn  
PING upiter.sina.com.cn (202.112.8.5) 56(84) bytes of data.
```

```
[webg@BigPc]$ ping ent.sina.com.cn  
PING upiter.sina.com.cn (202.112.8.2) 56(84) bytes of data.
```

```
[webg@BigPc]$ ping ent.sina.com.cn  
PING upiter.sina.com.cn (202.112.8.3) 56(84) bytes of data.
```

多个域名对应一个 IP。

```
[webg@BigPc]$ ping cul.sina.com.cn  
PING upiter.sina.com.cn (202.112.8.3) 56(84) bytes of data.
```

```
[webg@BigPc]$ ping ent.sina.com.cn  
PING upiter.sina.com.cn (202.112.8.3) 56(84) bytes of data.
```

3) 可能是一个站点有多个域名对应,例如, www.pku.edu.cn 和 www.pku.cn 等价, net.cs.pku.edu.cn 和 net.pku.cn 等价。

如何解决由于后三种情况造成的重复搜集呢?

注意不能简单的根据 IP 地址来判断是否为同一个站点,因为有虚拟主机的情况。要解决重复收集网页,就要找出那些指向同一物理位置 URL 的多个域名和 IP。这是一个逐渐累积的过程。首先要累积到一定数量的域名和 IP, 比如 100 万个, 然后把这些域名和 IP 对应的首页和首页链接出的最开始的几个页面抓取回来。如果比较结果一样, 则应该归为一组。以后搜集的时候可以只选择其中的一个进行搜集。选择的时候应该优先选择有域名的, 有的网站对于直接用 IP 访问是被禁止的, 例如, www.163.com 对应 IP 地址为: 202.108.42.73, 202.108.42.91, 202.108.42.64, 202.108.42.63, 202.108.42.71, 202.108.42.72。但是直接用 http://202.108.42.73/ 访问是被拒绝的。

在 TSE 中没有实现此功能, 有兴趣的读者可以自己练习实现。

## 第五节 搜集信息的类型

搜集信息类型取决于最后搜索引擎提供服务需要哪些信息,只要把最后提供的信息搜集回来,搜集子系统的目的就达到了。

在 TSE 中不仅限于 HTML 文本和普通文本,还可以搜集 pdf, doc 等资源,因为这些资源可以利用免费得到的转化工具如 pdftotext<sup>①</sup>, wvware<sup>②</sup> 转化为文本格式。根据 MIME 规范(RFC2045 et al. 2004), TSE 搜集的网页类型包括:“text/html”, “text/plain”, “text/xml”, “application/msword”, “application/pdf”, “application/postscript”, “application/vnd. ms-excel”, “text/rtf”, “application/vnd. ms-powerpoint”。

在 TSE 搜集网页文件类型内,还具有过滤 URL 功能,该函数在文件 Page. cpp 中实现。

```
/*
 * 过滤掉无用的链接
 * 如果要过滤,返回 true; 否则返回 false
 */
bool CPage::IsFilterLink(string plink)
{
    if( plink.empty() ) return true;
    if( plink.size() > URL_LEN ) return true;

    const char * filter_str[] = {
        "gate", "search", "library", "data/scop", "uhtbin", "staff/staff",
        "enter", "userid", "pstmail?", "pst?", "find?", "ccc?",
        "fwd?", "tcon?", "& amp", "Counter?", "forum", "cgisirsi",
        "+", "{", "}", "proxy", "login", "mailto:", "javascript:"
    };

    int filter_str_num = 25;
    string link = plink;
    CStringFun::Str2Lower( link, link.length() );

    for(int i=0; i<filter_str_num; i++){
        if(link.find(filter_str[i]) != string::npos)
            return true;
    }
```

① <http://poppler.freedesktop.org>

② <http://wvware.sourceforge.net>

```
    }  
  
    return false;  
}
```

为加快搜集过程,在搜集过程中,如果遇到 URL 对应的网页内容超过 5MB,搜集程序放弃该页面。这一过程在获取了网页头信息后(如第二节第三部分所述),根据其中指示的网页大小属性值,可以达到目的。

## 第六节 小 结

本章首先介绍了超文本传输协议,为搜集工作提供必要的背景知识;接着给出我们的搜索引擎教学程序 TSE 的系统结构,结合 TSE 的搜集端的源程序进行讲解,分析了一个综合性搜索引擎搜集端所应具备的绝大部分功能。

本章内容是搜索引擎三个步骤(网页搜集、预处理、查询服务)的首要环节。对于本章内容的理解,是深入理解整个搜索引擎流程的基础,同时考虑到预处理和服务模块,会更清楚为什么在搜集这个环节需要保存原始网页库和网页结构库。

## 第四章 对搜集信息的预处理

在第三章中,结合 TSE 搜索的 C++ 源程序代码介绍了搜索引擎工作流程中的网页搜集部分。本章开始讲述流程中的网页预处理和索引部分。

各节内容安排如下:首先给出信息预处理的系统结构,继而介绍建立索引网页库的算法,网页编码识别,中文切词技术,针对网页的分析,最后讲如何生成用于查询的网页倒排索引文件。

在 TSE 中网页预处理对应第三章图 3-4 的中间部分,我们将它单独取出来,去掉 TSE 中没有实现的 PageRank 部分后,如图 4-1 所示。

经过 TSE 的 Web 信息搜集,保存下来的网页信息是按照天网格式顺序存储的。按照天网格式保存网页信息,容错性好,即使有数据损坏,也是局部性的,不会导致扩散或者其他数据无法存取。缺点是不能够按照网页 URL 随机存取其所指向的网页。因此网页预处理的第一步就是为原始网页建立索引,实现图 4-1 中索引网页库,有了索引就可以为搜索引擎提供网页快照功能(在本章第一节中讲解);接下来针对索引

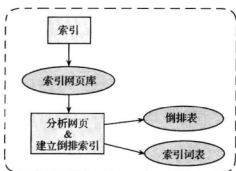


图 4-1 网页预处理系统结构

网页库进行网页切分,将每一篇网页转化为一组词的集合(在本章第三节中讲解);最后将网页到索引词的映射转化为索引词到网页的映射,形成倒排文件(包括倒排表和索引词表),同时将网页中包含的不重复的索引词汇聚成索引词表(在本章第四节中讲解)。

网页预处理处于搜索引擎三段式工作流程的中间,所产生的数据都是中间数据,如果不提供必要的应用程序接口,难以作为数据产品提供给其他程序使用。

### 第一节 索引网页库

使用 TSE 的 Web 信息搜集程序,获得的网页信息是原始网页库,取其中一个文件 Tianwang.raw.2559638448,来进行下面的讲解,它包括 12 933 个网页,容量为 146MB。文件名 Tianwang 表示存储格式符合天网格式要求,扩展名 raw 是表示未经过处理的原始网页,扩展名 2559638448 是 TSE 搜集程序中线程号为

2559638448 搜集器搜集到的原始网页信息。注意:在 32 位文件系统中,每个文件不能超过 2GB,负责存储网页信息的线程要有检查机制,超过后应该存储到新创建的文件中。天网格式存储网页支持网页压缩存储,在实际的面向海量数据的搜索引擎中是很必要的,因为网页信息通常可以压缩到原来所占空间的 1/3 以下,节省大量磁盘空间开销。

在天网格式中一个原始网页库由若干记录组成,每个记录包括记录头部信息(HEAD)和数据(DATA),每个数据由网页头信息(header),网页内容信息(content)组成,如图 4-2 所示,其中具体内容的含义请参照第三章第二节网页信息存储的天网格式。索引网页库的任务就是完成给定一个 URL,在原始网页库中定位到该 URL 所指向的记录。

```
version:1.0           //记录头部信息(HEAD)
```

```
url:http://net.pku.cn/~cnds/
```

```
origin:http://net.pku.cn/~cnds
```

```
date:Tue,16 Sep 2003 14:13:20 GMT
```

```
ip:162.105.203.25
```

```
length:11683
```

```
HTTP/1.1200 OK       //网页头信息(header)
```

```
Date:Tue,16 Sep 2003 14:19:15 GMT
```

```
Server:Apache/2.0.40(Red Hat Linux)
```

```
Last-Modified:Tue,16 Sep 2003 13:18:19 GMT
```

```
ETag:"10f7a5-2c8e-375a5cc0"
```

```
Accept-Ranges:bytes
```

```
Content-Length:11406
```

```
Connection:close
```

```
Content-Type:text/html;charset=GB2312
```

```
XXXXXXXXXX          //网页体信息(body)
```

```
.....
```

```
XXXXXXXXXX
```

图 4-2 原始网页库中的记录格式

如果不对网页库建立索引信息,可以通过顺序查找的方法完成 URL 到指定记录的过程,但是会消耗大量的 I/O,数据量增大的时候不能够满足搜索引擎的快速响应要求,所以需要创建索引。对原始网页集 R,索引网页库算法描述如图 4-3 所示。

```

a. 初始化,文档编号 id=0
b. for each  $r_i$  in R do
  begin
    b1. 读取  $r_i$  的记录头部和数据 read( $r_i$ ),
      得到 URL,  $r_i$  在 R 中偏移位置 offset 和网页内容 content
    b2. 记录索引信息 write(R,  $r_i$ )
      b2.1 记录 id, offset 和网页内容摘要 digest(content)到网页索引文件
      b2.2 记录 URL 摘要 digest(URL)和 id 到 URL 索引文件
      b2.3 id 加 1
  end
end

```

图 4-3 索引网页库算法

索引网页库算法在程序 DocIndex.cpp 中实现。搜索引擎面对的是海量数据,选择数据结构的时候需要考虑两个因素:紧凑的数据格式和高效的检索能力。在 TSE 中,对网页内容和 URL 的摘要算法可以采用 MD5 算法,产生 16 字节的唯一标识。为了便于查看,把 16 字节的标识转化为可以用 ASCII 码表示的 32 字节唯一标识。

网页索引文件 Doc.idx 如表 4-1 所示,以 ISAM(索引顺序访问模式)存储。这种结构可以保证数据的紧凑性和  $O(1)$  的检索能力。为节省空间,索引文件中的每一行记录不保存文档的长度,因为文档长度可以通过后续文档起始位置偏移和当前文档起始位置偏移的差获得。为了保证对最大文档号数据读取的一致性,在最后一行增加“哨兵”,它不对应实际的文档数据,其中文档摘要为空,表示文档索引的结束。

URL 索引文件 Url.idx 如表 4-2 所示,以 ISAM 存储,包含了 URL 的摘要和文档编号。为了能够快速的对给定 URL 找到对应的文档编号,将表 4-2 按照 URL 摘要排序,得到文件 Url.idx.sort。然后根据二分查找算法在 Url.idx.sort 中查找到对应的文档编号。采用二分查找算法获得网页快照的实现在 Snapshot.cpp 文件中。

表 4-1 网页索引文件

文档编号	偏移位置	网页内容摘要
0	0	bc9ce846d7987c4534f53d423380ba70
1	76760	4f47a3cad91f7d35f4bb6b2a638420e5
2	141624	d019433008538f65329ae8e39b86026c
.....	.....	.....
12931	210383421	e109b8aa7833faeca510d7619a214544
12932	210411107	

表 4-2 URL 索引文件

URL 摘要	文档编号
5c36868a9c5117eadbda747cbdb0725f	0
3272e136dd90263ee306a835c6c70d77	1
6b8601bb3bb9ab80f868d549b5c5a5f3	2
.....	.....
7bf8caae9e9b69ebcdc2b1a09dfdb43	12931

## 第二节 网页编码识别

网页数据作为搜集信息预处理的输入,需要解决的问题是把网页中的字节序列转化为字符序列,只有在字节序列上应用正确的编码,才能识别出有意义的字符序列。如果网页是采用 ASCII 编码方式书写的,这个过程很简单。但是如果采用各种单字节或多字节的编码方案,如 Unicode 的 UTF-8 编码,不同的国家或供应商的特定标准编码,事情就变得复杂了。网页编码识别可以采用基于机器学习的分类方法来解决,但通常采用“启发式方法”、“用户选择”或者从获得的文档元数据中获得。

作为编码识别输入的网页文档,有可能以 zip 形式压缩,以 Microsoft Word 的 doc 形式或 Adobe 的 pdf 形式输入。此处假定这些都已经转化为 txt 的文本形式,因为我们关注的重点是如何识别文本形式字节序列的编码。在正式开始编码识别前,需要了解三个基本而重要的概念,然后熟悉一些常用字符编码。在所获取的网页中,网页主要是由两种语言书写的,即英文和中文。英文相对来说容易识别,所以我们侧重于分析中文网页编码的识别。最后讲解在开发中文搜索引擎中主要需要识别的几种中文编码,并给出实现代码。

### 一、基本而重要的概念

字符集(character repertoire):一组不同字符的集合。既然是集合,那么这些字符之间没有顺序,它只是定义了字符的形状和名称。通常一个字符集是一个国家或地区,或者某种语言文字中使用的所有符号的集合。简单地说,英语国家的字符集是  $C=\{a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W,X,Y,Z\}$ ,一些标点符号。中国的字符集就是“汉字”和标点。

什么是一个字符?事实上,越是基本的概念往往越难以定义。Unicode 标准中一个比较抽象的定义是“用来组织,控制和表示文本数据的信息单元”,而且事实上不一定如此,只能说它是大家约定俗成的,或者由标准化的人根据自己的判断来鉴别,我们只能接受。要注意,①“长得像”的字符不一定是同一个字符,如拉丁字符集中大

写字母 A 和希腊字符大写阿尔法(alpha),也就是说不能相信你的眼睛。②为了标识字符,Unicode 标准给每个字符一个名称,名称得用文字表示出来,为了避免问题的循环讨论,规定字符的名称只能使用字母 A~Z 及空格和连字符,如上面所说希腊字符的大写阿尔法,学名叫 Greek uppercase alpha。③字符的定义有一定的人为因素,如在一个国际化的字符集如 Unicode 中,有时虽然属于不同语言的字符,但是“长的太像了”,我们就规定它是同一个字符。又如字符 ú 在 Unicode 中是一个单独字符,但若规定它是一个字符 u 加一个读音字符也未尝不可。

字符编码(character code):一个从字符集到一个非负整数集合直接的一一映射,经常用一个表格给出。通常似乎是译为“编码”,但我们更倾向于用“代码”,以防和 character encoding 混淆。其实就是让无序的字符集变成有序序列,也就是给字符集中每个字符一个编码位(术语 code position,有时写作 code number, code value, code element, code point, code set value, 或者仅用 code)。注意,作为映射值域的非负整数集合不一定是连续相邻的非负整数,可以有“空洞”。另外,字符编码还包括对一些控制字符(control character)的映射,这些控制字符不属于术语字符集的范畴,但它们在字符编码中有相应的编码位,如 ASCII 字符编码中换行(linefeed)的编码位为 10(十进制)。

字符编码算法(character encoding):把字符对应的字符编码位映射为八位组序列的方法,也就是把一定范围内(依字符集大小而定)的非负整数映射为字节序列的算法。

让问题显得混淆的关键是人们对这些术语的滥用。①Character code 常常隐含定义了字符集(不严格的,我们可以把字符集看成字符编码的“定义域”中所有可打印(printable)字符的集合),且 character code 常常又隐含了默认的字符编码算法。②Character set 是一个不标准的术语,非常容易混淆,不推荐使用,因为从字面上看它应该指字符集,但它常常又指字符编码,并且隐含字符编码算法。不幸的是 Internet 官方标准的 MIME 头中使用 charset(这是个生造的词),实际表示字符编码算法。③虽然通常来说,每种语言有对特定字符集的要求,不仅如此,语言的设置和字符问题是两个截然不同的话题。有时一些应用程序的语言(language)选项,可能指的是字符集,字符编码或者字符编码算法。例如英文版 IE 浏览器中用 languages(中文版翻译时较为正确的译作“编码”)选项来选择字符编码算法。④还有一些应用程序把字体(font)和上面的概念混淆起来,问题简直是一团糟。一组字形的集合组成某种字体,从实现上讲,一种字体就是一组编了号的字形(从计算机图形学的角度,字形可以用一个位图或者矢量数据表示)的集合。编号应该对应它表示的相应字符(某字符集的)在特定字符编码中的编码位,所以说字体是和字符代码相关的(字符编码又隐含了字符集)。本质上说,字体涉及字符在计算机或者打印机等设备上的显示,通常过程是这样的:如果想显示一个字符,首先要选定一个字符集,然后选择一个对这个字符集的编码方案,更常见的是直接选择一个字符编码(因为字符编码隐含了字



符集,如“GB2312”),然后根据这个字符编码给定一个编码位,窗口系统(如 X-Win-dow)或操作系统(如 Microsoft Windows),根据这个编码位查找字体库中当前设置的或默认的字体中对应的一个字形,用图形学的方法显示出来。

这部分内容的英文资料,可以参看 A tutorial on character code issues, <http://www.cs.tut.fi/~jkorpela/chars.html>。

## 二、常用字符编码

计算机只能处理由 0 和 1 组成的数据,要把对人友好的字符映射到二进制串,需要考虑进行字符编码。常用的字符编码有 ASCII、Unicode、GBK 等,下面开始逐一介绍。

### 1. ASCII

ASCII(American Standard Code for Information Interchange)是最为常用的字符编码,它是一种七位编码(编码位 0~127)。ASCII 字符编码我们都很熟悉,其细节就不赘述,但有几点要注意。ASCII 最初出现在 20 世纪 50 年代,之后出现了很多变体。最初的版本现在常称为 US-ASCII,官方标准是 ANSI X3.4-1986。另外,ISO646 定义了和 US-ASCII 类似的字符编码,但它把 US-ASCII 中对应字符@[\|} 的编码位定义为可以用本民族字符自定义的位,而它同时给出的国际参考版(International Reference Version)则和 US-ASCII 相同。大多数字符编码都把 US-ASCII 作为子集并且编码位也相同,这样无论怎样进行编码转换,使用 US-ASCII 一般都是安全的。由于以上原因,传输中最安全的字符包括:空格,大写字母 A~Z,小写字母 a~z,以及! "% &.( ) \* +, - . / ; : < = > ? 这 19 个字符。

计算机教材中有个广泛使用的词 ASCII 文件(ASCII file),实际指的是任何非二进制的文本文件,其实它使用的不一定 ASCII 编码。

### 2. ISO8859 字符编码族

ISO8859 编码是针对拉丁语系的字符对 ASCII 的扩展,包括 ISO8859-1~ISO8859-16(但 ISO8859-12 没有定义),它们都是 8 位编码。不同的 ISO8859 编码针对不同的语言使用不同的方法和字符对 ASCII 做了扩展。例如,ISO8859-1 是针对西(北)欧国家语言的,ISO8850-2 是针对中(东)欧国家语言的。但它们有共同的特征:0~127 编码位与 ASCII 编码相同,编码位 128~159 没有使用(为控制字符预留),160~255 则是根据不同语言的字符扩展。

目前,ISO8859-1 经常作为很多应用程序默认的字符编码,它的编码算法通常用 ISO-8859-1 表示。

### 3. 一些厂商的 8 位字符编码

微软公司在 DOS 操作系统时代使用的编码是 DOS 字符编码(DOS character codes),或者叫编码页(Code Pages,CP)。它也是对 ASCII 的扩展(0~127 编码位相同),但是与 ISO8859 字符编码族的扩展方法完全不同。不同的编码页表示不同的编码,如最早的美国英语的编码页是 CP 437。

微软公司 Windows 操作系统使用的编码是 Windows 字符集(Windows Character Sets),也是一系列编码页。它们多数和 ISO8859 字符编码族很相像,都兼容扩展了 ASCII(0~127 编码位相同),160~255 编码位的字符与某个 ISO8859 字符编码相同,只是编码位 128~159 有一些可打印(printable)字符,而非控制字符,如 windows-1252 和 ISO-8859-1 类似。

苹果公司的所有产品都使用统一的一个 8 位字符编码:Macintosh 字符编码(Macintosh character code)。它也是对 ASCII 的扩展(0~127 编码位相同),但编码位甚至字符集都和 ISO8859-1 不同。

IBM 大型机(mainframes)使用的是 EBCDIC 字符编码,也是 8 位字符编码,它包括 ASCII 中的所有字符,但是编码位则和 ASCII 不兼容。

### 4. Unicode

ISO/IEC 10646 定义了一个国际字符集,叫通用字符集(Universal Character Set, UCS),并且定义了所有字符的编码位。它试图包括世界各国语言使用的所有字符和符号,并且还在不断扩充,它甚至定义了 32 位的编码空间。ISO/IEC 10646 对中日韩文字实施统一编码,叫做“中日韩统一编码汉字”(CJK Unified Ideographs)。CJK 是“中日韩汉字”的简称,详细一些的写法是:Chinese Hanzi, Japanese Kanji and Korean Hanja。其中汉字的来源有《辞源》、《辞海》、《汉语大词典》、《中国大百科全书》、《四库全书》和方正排版系统。

Unicode 是 Unicode 协会(Unicode Consortium)对 ISO10646 的一个实现(字符集,字符编码),并且它为了保证不同平台和应用的统一性,增加了一些限制,如它规定了字符编码算法(见后续部分)。目前 Unicode 使用 16 位字符编码。

### 5. 中文字符编码

#### (1) GB2312-80

GB2312 编码(GB2312-80),中华人民共和国国家标准汉字信息交换用编码,是由中华人民共和国国家标准总局发布的一个关于简化汉字的编码,通行于中国大陆地区及新加坡,简称国标码(GB 就是国标的汉语拼音首字母)。它是一个 16 位的字符编码,是最常用的中文字符编码和字符集。

该编码的字符集收录常用汉字 6763 个,图形符号 682 个,它规定所有汉字和图

形符号组成一个  $94 \times 94$  的矩阵,在此方阵中,每行成为一个“区”,每列称为一个“位”,区码和位码分别需要 7 个二进制位表示,为了方便分别用一个字节表示,所以一个 GB2312 编码的汉字需要两个字节。国标码又叫“区位码”。

国标码和 ASCII 编码的英文字符可以混用,通常用字符中冗余的最高位来标识,最高位为 0 时表示是 ASCII 码,最高位为 1 时表示是 GB2312 编码的高位字节(区码)或者低位字节(位码)。两个字节中,第一个字节(高字节,区号)加 32(20H),第二个字节(低字节,位号)加 32(20H),用这两个值来表示一个汉字的编码位。

在 GB2312 中,汉字区的“高位字节”的范围是  $0xB0-0xF7$ ,”低位字节”的范围是  $0xA1-0xFE$ ,占用的码位是  $72 \times 94 = 6768$ 。其中有 5 个空位是  $D7FA-D7FE$ ,共定义 6763 个汉字(可参看网址 [http://zh.wikipedia.org/wiki/GB\\_2312](http://zh.wikipedia.org/wiki/GB_2312))。

## (2) Big5

Big5,又称为大五码或五大码,是使用繁体中文社群中最常用的计算机汉字字符集标准,共收录 13060 个汉字。中文码分为中文内码及中文交换码两类,Big5 属中文内码,知名的中文交换码有 CCCII、CNS11643。Big5 虽普及于中国台湾、香港与澳门等繁体中文通行区,但长期以来并非当地的国家标准,而只是业界标准。倚天中文系统、Windows 等主要系统的字符集都是以 Big5 为基准,但厂商又各自增删,衍生成多种不同版本。2003 年,Big5 被收录到中国台湾官方标准的附录当中,取得了较正式的地位。这个最新版本被称为 Big5-2003。

Big5 码使用了双字节的存储方法,以两个字节来安放一个汉字。“高位字节”使用了  $0x81-0xFE$ ,”低位字节”使用了  $0x40-0x7E$ ,及  $0xA1-0xFE$ (可参看网址 <http://zh.wikipedia.org/wiki/%E5%A4%A7%E4%BA%94%E7%A2%BC>)。

Big5/3  $0x8140-0xA0FE$  保留给使用者自定义字符(造字区)

Big5/1  $0xA440-0xC67E$  常用汉字,先按笔画再按部首排序。

Big5/2  $0xC940-0xF9D5$  次常用汉字,也是先按笔画再按部首排序  
 $0xF9D6-0xFEFE$  保留给使用者自定义字符(造字区)

## (3) GBK

GBK 编码是由前电子部科技质量司和国家技术监督局标准化司于 1995 年 12 月颁布的指导性规范,注意,它不是国家标准。GBK 的 K 是“扩展”的汉语拼音第一个字母。GBK 与国家标准 GB2312 兼容,即包括所有 GB2312 字符集且编码位相同。同时,它包含 Unicode 1.1 和 GB 13000.1-93 的 CJK 汉字集(20902 个,但编码位不同)。对于符号,GBK 除了包含 GB2312-80 和 GB12345-90(信息交换用汉字编码字符集第一辅助集)中包括的全部非汉字符号外,还涵盖我国台湾地区中文标准交换码 TCA-CNS 11643-92(与其对应的内码为 Big5)中的绝大多数符号。

从 Windows 95 中文版起,Windows NT 3.51, 4.0, Windows 2000, Windows CE, Linux, Java 已经全面支持 GBK。

注意,尽管 GBK 支持 CJK 字符集,但它的编码位完全不同于 ISO/IEC 10646 和

Unicode。重要的是它对 GB2312 的兼容性扩展,它只是一个过渡性规范,起到了从 GB 2312 向 GB18030-2000(见后)过渡的作用。

GBK 是在兼容 GB 码的基础上,将 GB 码中不存在的 ISO 10646.1 汉字,按 Unicode 编码顺序进行重新编码。其编码空间包含几个子集,其中 GBK/2 就是 GB 码汉字;各子集的编码情况见下(可参看网址 <http://www.clyrics.com/stonec/hanzi/gbngbk.htm>)。

GBK/1 0xA1A1-0xA9FE 846 717

GBK/2 0xB0A1-0xF7FE 6768 6763

GBK/3 0x8140-0xA0FE 6080 6080

GBK/4 0xAA40-0xFEAF 8160 8160

GBK/5 0xA840-0xA9A0 192 166

#### (4) GB13000

GB 13000 是等效采用 ISO/IEC 10646 的中国国家标准版本。目前 GB 13000 是 1993 年版,对应 ISO/IEC 10646-1:1993。对应 ISO/IEC 10646-1:2000 的 GB 13000-1:2000 版本目前还没有出版。

#### (5) GB18030

国家标准 GB18030-2000《信息交换用汉字编码字符集基本集的扩充》是我国继 GB2312-1980 和 GB13000-1993 之后最重要的汉字编码标准,并且还增加了藏、蒙等少数民族的文字,是我国计算机系统必须遵循的基础性标准之一。GB18030 目前的最新版本是 GB18030-2005。GB18030-2005 与 GB18030-2000 的编码体系结构是完全相同的。

### 三、常用字符编码算法

有了字符集和字符编码,在实际传输和存储时怎么表示呢?这就涉及字符编码算法的问题。前面提到,字符编码算法就是把字符对应的字符编码位映射为八位组序列的方法。

最简单的算法就是“不编码”,用字符的编码位表示字符,当然需要根据字符集的大小确定一个固定个数的八位组表示一个字符。对于七位和八位的字符编码,最直接的方法就是用一个八位组直接表示一个字符(对七位编码,八位组最高位可以冗余或者提供校验),其实就是不用编码算法转换。如 ASCII,ISO-8859-1 等,一般来说根本不需要考虑编码问题。类似地,16 位编码同样可以不编码,也就是直接用它的编码位(整数)对应的 16 位二进制数表示字符,这个二进制数表示的八位组序列也就自然是这个字符的编码了。

特别的,对于 GB2312 和 ASCII 混合使用的场合(一个是 16 位编码——实际 14 位有效,一个是 7 位编码),简单的方法就是用八位组的最高位提供鉴别信息,如果最高位为 0,表示这个八位组是 ASCII 编码的字符;如果最高位是 1 表示这个八位组和

其后八位组是一个 GB2312 编码的字符。

Unicode 最基本的编码算法 UCS-2 和前面所述的同理,用两个相邻的八位组  $m$  和  $n$  表示编码位为  $256 \times m + n$  的字符。但问题是我们所处的是英语主导的世界语言,大量的现有文本资源是英语文献,把它们用 Unicode 编码算法来编码,需要的八位组序列长度(不管用来传输还是存储)将是用 ISO-8859-1 编码需要的八位组序列长度的两倍。这就会产生效率问题,但后者则没有国际化字符集的好处,很自然需要找到某种折中的方案。两种常用的编码方案是 UTF-8 和 UTF-7。

UTF-8(Unicode Transformation Form 8-bit form)是 Unicode 的一种变长字符编码算法。编码位小于 128 的字符(即 ASCII 字符)用一个八位组编码;其他的编码位用 2~4 个八位组的序列表示,且每个八位组的值为 128~255(即最高位为 1)。也就是说最高位为 0 的八位组直接表示一个 ASCII 字符,最高位为 1 的八位组表示一个字符编码序列的一部分。对应汉字在 Unicode 中所处的编码位(16 位),使用的编码是 3 个八位组:1110...10.....10.....,横线处组成的 16 位即字符所处的编码位。例如,汉字“你”的 Unicode 编码位为 0x4f60(0100111101100000),它的 UTF-8 编码八位组序列为 0xe4bda(11100100 10111101 10100000)。编码位值为 8~11 位的,编码为 2 个八位组:110.....10.....,编码位值 17~21 位的,编码为 4 个八位组:1 1 1 1 0...10.....10.....10.....。

UTF-7(Unicode Transformation Form 7-bit form)编码算法则把编码位用一个至多个值界于 0~127 的八位组(即只用八位组的 7 位)表示。大多数 ASCII 也同样用一个八位组直接表示,但是显然,某些八位组必须被保留用来表示随后的若干个八位组组成的整体代表一个字符的编码。

当用户面对一个八位组序列的时候,即使知道它是一段明文(非加密)文本,没有编码方式的信息也根本无法知道它代表的信息内容。Internet 协议中常使用 MIME 头指出数据的编码算法,如 Content-Type: text/html; charset=iso-8859-1,表示数据是 html 文本,字符编码为 ISO-8859-1。注意,这里 charset 实际表示字符编码算法。

HTML4.0 标准允许 HTML 文档使用 Unicode 的所有字符。但 URL 规范却把 URL 可以使用的字符限制在 US-ASCII 的一个子集里(详见 RFC1738)。所以 HTML 文档里的 URL 如果使用了这个子集之外的字符,都需要进行 URL 编码(URL encoding)。URL 编码算法是针对 ISO8859-1 编码(字符集)的,但实际可以把任何八位组序列当成是 ISO8859-1 编码的八位组(即无意义的字节流)进行处理。ISO8859-1 编码位中,对应 ASCII 码中的控制字符,URL 语法需要的保留字符(如 \$& ? : ; , @ = + /),非 ASCII 字符,及某些不安全的字符(如空格 < > { } ~ \ ^ [ ] )对应的编码位都要进行编码。编码算法是把每个这些需要编码的八位组转换成 3 个八位组:一个百分号“%”,后接 2 个表示这个八位组值(ISO8859-1 编码位)的十六进制数的 ASCII 码表示。例如,空格在 ISO8859-1 中的编码位为 32(十六进制 20),

它的 URL 编码为“%20”。中文字符“你”的 GBK 编码为 2 字节“C4E3”，则它的 URL 编码为 ASCII 字符串“%C4%E3”。这里把“C4”和“E3”分别作为两个 ISO8859-1 编码位处理。

#### 四、字符的输入和显示

用户从键盘上输入某个字符“你”，希望显示器上能正确显示“你”，存为磁盘文件时存的也是“你”，从打印机打印出来的是“你”，把它通过网络传输给别人，希望收信人也能看到“你”。但字符的输入和显示并不像我们相象的那么简单。

从本质上说，输入字符实际上是通过键盘在系统和应用程序内存中得到某个正确的字符编码位值。这个过程显然和字符集和字符编码有关系。

拉丁语系使用较少的字母作为字符，如 ASCII 字符编码。键盘最初也是为他们设计的，所以一般来说当敲击键盘的 a，就得到了 a 的 ASCII 编码位值。按住 Shift 键再敲击 a 就得到了大写字母 A 的 ASCII 编码位值，这是通过组合键输入字符的一个例子。使用稍大一点的字符集（如 ISO8859 字符族）的国家，如瑞典，键盘上可能有些特殊键以便于某些字符的输入，但这是从硬件一级的扩展。当然可以用软件的办法，如改变系统的键盘的键码到字符编码位的映射关系，这样也许得到的并不是敲击的那个键盘键上印着的字符，此时系统可能提供“可视软键盘”在屏幕上以方便输入识别。

像汉字这样使用大字符集的语言，它的字符输入显然不能和键盘键位一一对应。这时就需要某种“输入法”，输入法实际是把敲击的特定键位序列映射为一个字符的编码位的软件，如“拼音输入法”，“五笔输入法”等。例如，用拼音输入法，键入“ni”并用数字键选择，在内存里得到了“你”的 GB2312 编码（16 位）。

字符的显示则是把内存存储中八位组序列表示的字符编码位转换成正确字符的字形，显示在屏幕上，这个过程在前面讨论过。可以看出，显示和输入是两个相对独立的过程，显示只需要系统能正确识别字符编码并且有相应字符集的字体系，输入则需要安装输入法。如有的 Linux 系统可以正确显示汉字，但是不能输入汉字。

一个简单的流程如图 4-4 所示。

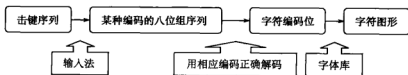


图 4-4 字符的输入和显示流程

另外，需要说明一点，Unicode 字符可以用“U+nnnn”来指代，“nnnn”是字符的 Unicode16 位编码位的十六进制字符表示。如 HTML 文档中，可以用“& # number;”来指代一个字符，number 为这个字符在该字符编码中的编码位。

## 五、编码识别

在网页预处理阶段,对于中文网页,需要切词。通常切词软件只接受一种字符编码,所以用户需要首先识别输入网页的编码,然后转为切词软件所需的字符编码。这个过程的难点在网页编码识别。通常采用的网页编码识别过程是:①对于网页资源,从 http 返回的 head 信息中有 charset 域,则可以得到;②如果没有,则由页面 html 的 head 中的 meta 标签的 charset 属性决定;③仍然没有,则只能从此页面资源面向的用户语言类型来推测一个合适的 charset。前两步容易实现,下面介绍最后一步推测 charset 的实现过程。

针对中文网页常见的三类主要编码 GB2312, Big5 和 GBK,我们采用如下方法来识别。首先根据前面讲述的字符编码特征,把三类编码分布图画出来。其中 GB2312 区域是 B0A1-F7FE。GBK 区域包括 GBK/2, GBK/3 和 GBK/4 三个大的汉字区域,如图 4-5 所示。Big5 三个区域对应的是常用汉字、次常用汉字和造字区。

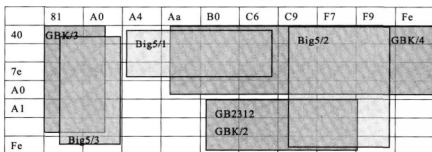


图 4-5 GB2312, Big5 和 GBK 字符编码分布

GBK 字符集比 Big5 和 GB2312 都要大。这样,标记的是 GBK,但是实际上是 Big5 的就无法判断了。根据编码的一些特征,GBK 有一部分字符 Big5 里没有,这部分就是我们用来判断是 GBK 而不是 Big5 的根据。而对于 Big5,它有一部分字符是常用字符集,这部分刚好和 GB2312 交叉很少,我们就取了这部分字符集作为判断是 Big5 而不是 GB2312 的根据。对于 GBK 和 GB2312,在后续处理上没有区别。

检查网页源码,判断 charset 的原则:

1) 如果网页中有 GBK 独有的字符(不属于 Big5,也不属于 GB2312),则 charset = GBK;

2) 如果网页中在 Big5 码常用字符集范围内,并且不在 GB2312 范围内的字符比例较大,则 charset = Big5;

3) 否则 charset = GBK。

当然还可以采用这样的原则来判断:①出现了非 Big5 中的字符,且在 GBK 中,则 charset = GBK 不用区分 GBK 和 GB。②如果字符都在 Big5 范围内,则当在

Big5/1(Big5 的常用字)内的比率高时,charset=Big5;else charset=GBK。

在天网搜索引擎的早期版本中,没有针对 UTF-8 编码的识别,这类网页由于切词软件不能识别,全部丢弃了,检索时就不会返回。现在的 Web 中,UTF-8 编码的网页越来越多,所以我们增加了判断,也将其转换为切词软件能够识别的字符,检索时就可以返回了。网页编码识别的实现源码在 <http://sewm.pku.edu.cn/src/paradise> 的分析与索引模块的 cconv 部分。其中关于简繁体转换和其他编码的转换是利用网上找到的头文件和库(Network Chinese Filter (NCF) 1.0,和 Hanzi Converter 的 hclib.h)改动的,时间比较久了,但考虑编码规范应该还是不会变的,因此还在使用。添加了 UTF-8 编码的判断和转换部分,转换部分直接利用 Linux 下的 iconv 库。之所以没有把各种编码转换都利用 iconv 来完成,是因为 iconv 是基于表转换的,根据我们的经验,对简体字的转换不如原来的库做的好。这样一来编码转换的问题在于正确识别出输入文档的编码,前面介绍了 GBK 和 Big5 编码的识别,下面讲解 UTF-8 编码的识别。根据 RFC2639(<http://www.ietf.org/rfc/rfc3629.txt>)UTF-8 标准中第 3 部分 UTF-8 的定义,可以理解 utf8\_gb.c 源程序中的 is\_valid\_utf8 函数。

每个使用 UTF-8 储存的字符,除了第一个字节外,其余字节的头两个位都是以“10”开始。

Char. number range (hexadecimal)	UTF-8 octet sequence (binary)
0000 0000-0000 007F	0xxxxxxx (00-7f)
0000 0080-0000 07FF	110xxxxx 10xxxxxx (c0-df) (80-bf)
0000 0800-0000 FFFF	1110xxxx 10xxxxxx 10xxxxxx (e0-ef) (80-bf) (80-bf)
0001 0000-0010 FFFF	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx (f0-f7) (80-bf) (80-bf) (80-bf)

在 is\_valid\_utf8 函数中的算法不是 UTF-8 字符串成立的充要条件,而是必要条件,所以某个条件不满足就返回 false。2/3/4 字节编码除了首字节以外,其他字节的值域都在 0x80-0xBF。

Is\_valid\_utf8 函数首先根据首字节值确定组成该字符的序列有几个字节。在 RFC2639 中给的例子编码最大长度是 4 字节,是针对 16 位的 BMP(Basic Multilingual Plane)字符,但实际上 Unicode 是 32 位编码,理论上对其编码的最长形式应该是 6 字节,所以 UTF-8 字节序还应该下面这两种字节序:

```
1111 10xx 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxx
1111 110x 10xx xxxx 10xx xxxx 10xx xxxx 10xx xxxxl0xx xxxx
```

为了和 UTF-16 的编码空间一致,在最新的 ISO 10646 标准中,最多只使用 4 字节编码,5 字节及 6 字节 UTF-8 已不再使用。



### 第三节 中文自动分词

对索引网页库信息进行预处理包括网页分析和建立倒排文件索引两个部分。中文自动分词是网页分析的前提。文档由被称为特征项的索引词(词或者字)组成,网页分析是将一个文档表示为特征项的过程。在提取特征项时,中文信息又面临了与英文信息处理不同的问题。中文信息和英文信息有一个明显的差别:英语单词之间用空格分隔;而在中文文本中,词与词之间没有明显的分隔符,中文词汇大多是由两个或两个以上汉字组成的,并且语句是连续书写的。这就要求在对中文文本进行自动分析前,先将整句切割成小的词汇单元,即中文分词(或中文切词)。用具体例子来说明,就是如何把“我的笔记本”这样连续书写的语句切分为“我”、“的”和“笔记本”三个词汇单元。

在检索和文档分类系统中,自动切词系统的速度直接影响整个系统的效率。中文信息的检索主要有两种:基于字的检索和基于词的检索。基于单字的检索系统建立单字索引。在检索时得到每个单字的索引,而后加以适当的逻辑运算,得到检索结果。而基于词汇的检索系统对词汇建立索引,检索词汇时一次命中。TSE 系统采用基于词的索引,具有较快的速度和较高的准确性,同时能够减少索引信息对磁盘空间的占用。

本节从介绍切词软件的原理开始,讲到真正切词软件的实现。此切词软件中使用的基本词典包括 108 782 个词条及其对应词频。词典和切词源程序可以在(ChSeg 2003)获得,运行在 Red Hat Linux 8.0 中调试运行正常。

#### 1. 算法介绍

自动分词的基本方法有:基于字符串匹配的分词方法和基于统计的分词方法。更多关于语言学的知识可以参看北大计算语言研究所网站(<http://icl.pku.edu.cn>)的相关内容,本小节内容直接引自该网站提供的资料。

##### (1) 基于字符串匹配的分词方法

这种方法又称为机械分词方法,它是按照一定的策略将待分析的汉字串与一个充分大的词典中的词条进行匹配,若在词典中找到某个字符串,则匹配成功(识别出一个词)。

按照扫描方向的不同,串匹配分词方法可以分为正向匹配和逆向匹配;按照不同长度优先匹配的情况,可以分为最大或最长匹配和最小或最短匹配;按照是否与词性标注过程相结合,又可以分为单纯分词方法和分词与标注相结合的一体化方法。常用的几种机械分词方法如下:

- 正向最大匹配;

- 逆向最大匹配;
- 最少切分(使每一句中切出的词数最小)。

另外,还可以将上述各种方法相互组合,如将正向最大匹配方法和逆向最大匹配方法结合起来构成双向匹配法。由于汉语单字成词的特点,正向最小匹配和逆向最小匹配一般很少使用。一般说来,逆向匹配的切分精度略高于正向匹配,遇到的歧义现象也较少。统计结果表明,单纯使用正向最大匹配的误差率为 1/169,单纯使用逆向最大匹配的误差率为 1/245(这可能是因为汉语的中心语靠后的特点)。

对于机械分词方法,可以建立一个一般的模型,形式地表示为  $ASM(d, a, m)$ , 即 Automatic Segmentation Model。其中,

$d$ : 匹配方向, + 表示正向, - 表示逆向。

$a$ : 每次匹配失败后增加或减少字串长度(字符数), + 为增字, - 为减字。

$m$ : 最大或最小匹配标志, + 为最大匹配, - 为最小匹配。

例如,  $ASM(+, -, +)$  就是正向减字最大匹配法(maximum match based approach, MM),  $ASM(-, -, +)$  就是逆向减字最大匹配法(简记为 RMM 方法), 等等。对于现代汉语来说, 只有  $m = +$  是实用的方法。

## (2) 基于统计的分词方法

从形式上看, 词是稳定的字的组合, 因此在上下文中, 相邻的字同时出现的次数越多, 就越有可能构成一个词。因此字与字相邻共现的频率或概率能够较好的反映成词的可信度。可以对语料中相邻共现的各个字的组合的频度进行统计, 计算它们的互现信息。计算汉字  $X$  和  $Y$  的互现信息公式为

$$M(X, Y) = \lg \frac{P(X, Y)}{P(X)P(Y)}$$

其中,  $P(X, Y)$  是汉字  $X, Y$  的相邻共现概率,  $P(X)$ 、 $P(Y)$  分别是  $X, Y$  在语料中出现的概率。互现信息体现了汉字之间结合关系的紧密程度。当紧密程度高于某一个阈值时, 便可认为此字组可能构成了一个词。这种方法只需对语料中的字组频度进行统计, 不需要切分词典, 因而又叫做无词典分词法或统计取词方法。但这种方法也有一定的局限性, 会经常抽出一些共现频度高、但并不是词的常用字组, 如“这一”、“之一”、“有的”、“我的”、“许多的”等, 并且对常用词的识别精度差, 时空开销大。实际应用的统计分词系统都要使用一部基本的分词词典(常用词词典)进行串匹配分词, 同时使用统计方法识别一些新的词, 即将串频统计和串匹配结合起来, 既发挥匹配分词切分速度快、效率高的特点, 又利用了无词典分词结合上下文识别生词、自动消除歧义的优点。

词汇切分算法最重要的指标是准确, 在兼顾准确性的情况下也要考虑时间复杂性。下面具体介绍正向减字最大匹配法。

## 2. 正向减字最大匹配法

正向减字最大匹配法切分的过程是从自然语言的中文语句中提取出设定的长度字串,与词典比较,如果在词典中,就算一个有意义的词串,并用分隔符分隔输出,否则缩短字串,在词典中重新查找(词典是预先定义好的)。

算法要求:

输入:中文词典,待切分的文本  $d$ ,  $d$  中有若干被标点符号分割的句子  $s1$ , 设定的最大词长  $MaxLen$ 。

输出:每个句子  $s1$  被切为若干长度不超过  $MaxLen$  的字符串,并用分隔符分开,记为  $s2$ ,所有  $s2$  的连接构成  $d$  切分之后的文本。

该算法的思想是:事先将网页预处理成每行是一个句子的纯文本格式。从  $d$  中逐句提取,对于每个句子  $s1$  从左向右以  $MaxLen$  为界选出候选字串  $W$ ,如果  $W$  在词典中,处理下一个长为  $MaxLen$  的候选字段;否则,将  $W$  最右边一个字去掉,继续与词典比较; $s1$  切分完之后,构成词的字符串或者此时  $W$  已经为单字,用分隔符隔开输出给  $s2$ 。从  $s1$  中减去  $W$ ,继续处理后续的字串。 $s1$  处理结束,取  $T$  中的下一个句子赋给  $s1$ ,重复前述步骤,直到整篇文本  $d$  都切分完毕。

考虑到词典的条目较多,而且每切一个词都要比较,采用 STL 中的 `map` 容器 (Josuttis 1999) 作为存储词典的数据结构。`map` 容器采用的数据结构是“红黑树”。“红黑树”是一种较常用的查找效率较高的平衡二叉搜索树 (Cormen et al. 2001)。在实际应用中可以采用 `hash` 表数据结构存储获得更快速的查找。

算法的主程序流程如图 4-6 所示,切词程序 `CutWord` 如图 4-7 所示。其中  $Max$

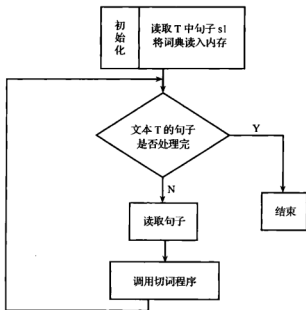


图 4-6 正向减字最大匹配算法流程

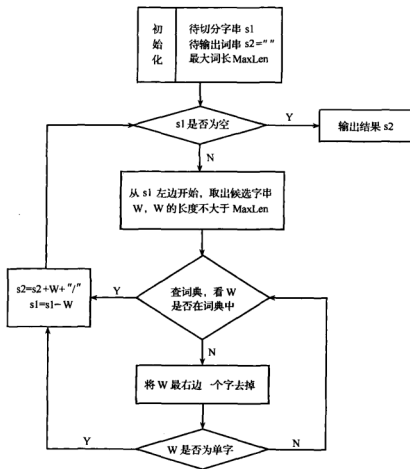


图 4-7 切词算法流程

Len 是一个经验值, 通常设为 8 字节 (即 4 个汉字),  $MaxLen$  过小, 长词会被切断; 过长, 又会导致切分效率低。

为进行算法分析, 先给出算法的伪码实现。

```

void SegmentAFile (T)                                // 对文件进行分词处理的函数
1   while( getASentenceFromFile (T, s1) )           // 循环读入文件中的每一行
2       s2 = CutWord (s1)                             // 调用句子分词处理函数
3       OutputSentence ( s2 )                          // 将分词结果写入目标文件

string CutWord ( s1 )
4   Preprocess ( s1 )                                  // 跳过非汉字部分字符串
5   While (s1 != "")                                   // 如果输入不为空

```

```

6      W = s1.substr ( 0, MaxLen )           // 取等于最大词长的候选词
7      While( length(W)> 1 )
8          If( FindInRBTREE (W) = false)      //如果不是词并且不是单字
9              then W = W - 1                 // 将 W 中最右边一个字去掉
10         s2 = W + "/"?                       // 将找到的词用分隔符隔开
11         s1 = s1 - W                         // 去掉找到的词,继续分析
12     return s2

```

### 算法分析:

设文本文件含有句子的数目为  $m$ , 句子的平均长度为  $k$ , 词典的条目为  $n$ 。实际中  $m$  和  $k$  远远小于  $n$ , 整个算法复杂度中起决定作用的步骤在于  $n$  相关的语句。

行 1,  $O(m)$   
 行 2,  $O(k \lg n)$       //通过对第 4 到 12 行子程序 CutWord 的复杂度分析得知  
 行 3,  $O(1)$   
 行 4,  $O(1)$   
 行 5,  $O(k)$   
 行 6,  $O(1)$   
 行 7,  $O(1)$   
 行 8,  $O(\lg n)$   
 行 9,  $O(1)$   
 行 10,  $O(1)$   
 行 11,  $O(1)$   
 行 12,  $O(1)$

整个算法的时间复杂度为  $O(mk \lg n)$ 。

上面是对效率的分析, 考虑到切词的效果, 本算法还可采用 (Cormen et al. 2001) 中讲述的“回溯”思想作改进。除了上述从左到右切分一遍句子, 还从右到左切分一遍, 对于两遍切分结果不同的字符串, 用回溯法重新处理。例如“学历史知识”顺向扫描的结果是: “学历/ 史/ 知识/”, 通过查词典知道“史”不在词典中, 于是进行回溯, 将“学历”的尾字“历”取出与后面的“史”组成“历史”, 再查词典, 看“学”, “历史”是否在词典中, 如果在, 就将分词结果调整为: “学/ 历史/ 知识/”。

## 第四节 分析网页和建立倒排文件

为网页建立全文索引是网页预处理的核心部分, 包括分析网页和建立倒排文件。二者是顺序进行, 先分析网页, 后建立倒排文件 (也称为反向索引), 如

图 4-8 所示。

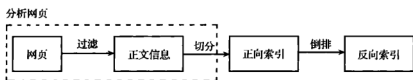


图 4-8 分析网页与建立倒排文件流程

### 1. 分析网页

分析网页过程如图 4-8 所示，它包括提取正文信息（指过滤网页标签，scripts, css, java, embeddedobjects, comments 等信息）和把正文信息切分为索引词两个阶段。形成的结果是文档号到索引词的对应关系表。每条记录中包括文档编号，索引词编号，索引词在文档中的位置信息，“索引词载体信息”（在本书中网页与文档是不加以区分的，但是网页中包含普通文档所没有的 HTML 标签信息，这些信息标识了文档中索引词的字体和大小写等信息，或称载体信息，是搜索引擎的服务阶段提供更好的结果排序所需要的）。在 TSE 中只保存了文档编号和索引词信息。

过滤网页中非正文信息的算法如图 4-9 所示。

```

a. 初始化, 指针 d 和 s 指向文档内容 C 起始地址
b. for each * s in C do
    begin
        b1. 如果 * s 是字符 '<', s 向后移动直到发现字符 '>', s 加 1
        b2. 否则 * s 赋值给 * d, s 和 d 分别加 1
    end
  
```

图 4-9 过滤网页中非正文信息算法

得到网页正文信息，调用切词模块，获得正向索引，格式如图 4-10 所示。每一个网页由两行信息组成，第一行是文档编号，第二行是使用切分模块将文档正文信息划分成索引词后的集合。

### 2. 建立倒排文件

搜索引擎面临大量的用户检索需求（几十～几千点击/秒），要求搜索引擎在检索程序的设计上要高效，尽可能地将大运算量的工作在索引建立时完成，使检索时的运算尽量的少。一般的数据库系统不能快速响应如此大量的用户请求，在搜索引擎中通常采用倒排索引技术。

```

0
中国/ 联通/ 郑州/ 分公司/.....
1
四平/ 风采/ 吉林/ 信息港/.....
.....
12726
逍遥/ 阅/ 建/ 站/ 日期/.....

```

图 4-10 正向索引表记录格式

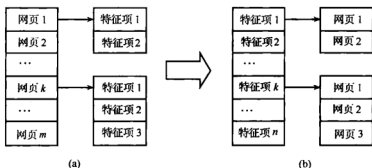


图 4-11 由正向索引建立反向索引

在图 4-8 中,创建倒排索引包括建立正向索引和反向索引。分析完网页后,得到以网页编号为主键的正向索引表,如图 4-11(a)所示(对应 TSE 中的正向索引如图 4-10 所示)。当反向索引建立完成后,得到图 4-11(b)。这是一个表的重组的过程,为了加快速度全过程需要在内存中完成。在小数据量时,有足够的内存保证该创建过程可以一次完成。数据规模增大后,可以采用分组索引,然后再归并索引的策略。该策略是,建立索引的模块根据当时运行系统所在的计算机的内存大小,将索引分为  $k$  组,使得每组运算所需内存都小于系统能够提供的最大使用内存的大小。按照倒排索引的生成算法,生成  $k$  组倒排索引。然后将这  $k$  组索引归并,即将相同索引词对应的数据合并到一起,就得到了以索引词为主键的最终倒排文件索引,即反向索引。具体程序实现可以参考 `CrtForwardIdx.cpp` 和 `CrtInvertedIdx.cpp` 的代码。

真正的搜索引擎,倒排文件很大,无法直接调入内存,通常在内存中存储以索引词和倒排表项偏移位置组合的 ISAM 信息。实现方法同样可以采用本章第二节介绍索引网页库的方法。

## 第五节 小 结

本章结合 TSE 讲解了索引网页库算法、中文切词算法、分析网页和建立倒排文件索引的方法。

本章内容作为搜索引擎运行三个阶段(网页搜集、预处理和查询服务)的中间环节,具有举足轻重的地位。对于本章内容的理解,可以加深对于整个搜索引擎流程的理解,同时考虑到网页搜集和查询服务模块,也会更清楚为什么这个环节需要考虑中文切词、分析网页和建立倒排索引等问题。



## 第五章 信息查询服务

经过第三章和第四章的学习,现在我们进入搜索引擎工作流程的最后一个环节——查询服务。查询服务包括接受用户输入的查询短语、检索、获得相应的匹配结果并显示给用户。此时我们已经有了索引网页库和倒排文件,需要做的就是通过查询代理实现索引数据与用户查询的互通。

本章首先给出查询服务系统结构,然后给出检索的形式化定义,最后结合 TSE 给出查询服务的具体实现。

在 TSE 中信息查询对应第三章图 3-4 的右侧部分,将它单独列出来,去掉 TSE 中没有实现的日志挖掘部分后,得到信息查询服务的系统结构,如图 5-1 所示。

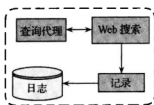


图 5-1 信息查询的系统结构

经过 TSE 的 Web 信息预处理,传递到服务阶段的数据包括索引网页库和倒排文件,倒排文件中包括倒排表和索引词表。查询代理接受用户输入的查询短语,切分后,从索引词表和倒排文件中检索获得包含查询短语的文档并返回给用户。

因为内存与外存(磁盘)的响应时间差距很大,在实际使用的搜索引擎中,为了提高响应时间,索引词表是驻留在内存中的,用户近期查询过的结果信息也是缓存在内存中的。如果内存足够大,所有倒排表项也可以驻留在内存中。只有这样,才能保证在大数据量和大访问量(如每秒 1000 个查询)的情况下,搜索引擎在秒级内得到响应。

### 第一节 检索的定义

首先,定义系统的元检索——单个词汇的检索方式。

设  $D = \{t_1, t_2, \dots, t_{|D|}\}$  为系统的特征项词典;集合  $P = \{p_1, p_2, \dots, p_{|P|}\}$  为系统当前保存的网页集合;系统的索引可表示为集合  $R = \{\langle t, p \rangle \mid r(t, p) > 0, \langle t, p \rangle \in D \times P\}$ ,其中,  $r(t, p)$  是相关度函数,表示词汇  $t$  和网页  $p$  的相关度,如果  $t$  是网页  $p$  的一个特征项,那么它就使用相关度算法给出相应的正值,如果不是,相关度为 0;搜索引擎系统  $S$  为一个三元组  $S = \{\langle t, p, r \rangle \mid \langle t, p, r \rangle \in D \times P \times R\}$ 。则有

$$WP(t) = \psi(t, S) = \{p \mid \langle t, p \rangle \in R, p \in P\} \quad (5-1)$$

其中,  $WP$  代表检索词汇  $t$  的相关网页集合,  $\psi$  函数是系统  $S$  的元检索函数。

显然,用户不可能总是进行词汇级的检索。大部分的用户输入的检索是词组或自然语句。如何从检索中提取出关键词,在各种信息检索系统中实现是不同的,有些

系统直接从检索输入中提取关键词;有些系统提取关键词后,再根据一些规则对关键词进行扩充。在本模型中,统一的将它们定义为关键词提取函数: $g(q, D) = \{t_1, t_2, \dots, t_m\}$ ,即通过函数  $g$  获得检索输入  $q$  的相关关键词集合。由公式(5-1),关键词  $t_i$  的相关网页集合为  $WP(t_i)$ ,则最终检索输入  $q$  的对应结果为

$$WP = f(WP(t_1), WP(t_2), \dots, WP(t_m)) = f(q, D, P, R) = f(q, S) \quad (5-2)$$

其中,函数  $f$  表示从  $q$  中提取的关键词的逻辑关系运算式, $f(q, S)$  是系统检索的抽象表达式。由公式(5-1)和公式(5-2)可以得到,作为文档特征项的关键词在检索过程中起到了桥梁作用,关键词选择的好坏直接影响到检索结果的质量。

## 第二节 查询服务的实现

### 一、结果集合的形成

根据用户输入的查询短语,产生结果集合,是检索倒排索引的过程。首先对用户输入的查询短语应用第四章中讲到的中文自动分词技术,获得查询  $q$  的向量表示, $q = \{t_1, t_2, \dots, t_m\}$ ,然后执行检索算法,算法描述如图 5-2 所示,实现代码在文件 TSE-Search.cpp 中。这个算法是实际搜索引擎检索算法的简化,应用在 TSE 中。实际搜索引擎的倒排索引中记录了索引词的权重和位置信息,检索阶段应该一起读出,并加以综合考虑;并且为了在获得结果前读取尽量少的数据,查询  $q$  中的  $t_i$  按文档频率的倒数降序排列。在 TSE 中,文档权值由索引词频率决定,并且选取结果  $K$  是包含查询词的所有文档。

```

a. 初始化,结果集合  $R = \Phi$ , 权值累加器  $A = 0$ 
b. for each  $t_i$  in  $q$  do
    begin
        b1. 读取  $t_i$  的倒排项数据 read( $t_i$ )
        b2. 逐项处理倒排项数据 merge( $R, t_i$ )
            b2.1 得到  $t_i$  的文档集  $\{d_u\}$ , 权值  $w_u$ 
            b2.2 执行布尔查询  $R = \text{Boolean}(R, \{d_u\})$ , 一般布尔运算为 AND
            b2.3 使用  $w_u$  更新结果集合  $R$  中元素对应的权值  $A$ 
    end
c. 根据累加器  $A$  从结果集合  $R$  中选取最大权值的  $K$  个结果 select( $R, A, K$ )

```

图 5-2 基本检索算法

图 5-2 中 b2.2 检索运算是算法中的重点,首先因为它的运算量很大。如果记  $|P(t_i)|$  为集合  $P(t_i)$  内所包含的网页数目,那么,它要处理的网页数目就是

$$NP(q) = \sum_{i=1}^n |P(t_i)|$$

其次,它处理的是集合运算。如果系统支持与和或运算,分别对应集合的交和并运算。如集合 A 和 B,做运算时需要将集合内的元素两两比较,这样的复杂度为  $|A| \times |B|$ ,即平方复杂度。显然,无法接受这么慢的速度,需要采用更加高效的算法。对于理论上提到的集合,在计算机中用表的方式表达。如果将这个表表示为有序表,集合运算就成了有序表的归并运算。有序表的归并运算的时间复杂度为  $O(|A| + |B|)$ ,这可以大大提高检索运算的效率。

如何将每个特征项对应的网页表达成按照网页编码顺序组成的有序表?这就需要排序。根据理论上的证明,无论采用何种排序算法,排序的最低复杂度为  $O(n \lg n)$ 。但是,在索引系统设计时,可以通过给予先搜集到的网页以小的网页编码的方法,使得索引项自动保持了顺序,避免这一步的运算。

通过以上的分析,可以得出从接收到用户的请求到得到相关的网页,相应的运算的复杂度为  $O(NP(q))$ ,即线性复杂度。

对检索得到相关网页集合,下面的工作就是计算每个网页和查询检索串  $q$  的相关度,它在图 5-2 所描述的基本检索算法的第 c 步完成。

## 二、查询结果显示

### 1. 列表显示摘要结果

用户界面主要用于和用户交互,包括响应用户的查询检索和记录用户的行为。

用户界面主要负责和用户直接接触的事件,如图 5-1 所示,它包括:

- 1) 获取用户的查询请求,提交给查询代理;
- 2) 查询代理检索索引词表和倒排表,产生结果按照一定的输出格式显示给用户;
- 3) 记录日志,包括用户查询短语和查询时间等信息。

对于功能 1),通过 HTML 语言的  $\langle \text{FORM} \rangle$  来实现。用户在相应的检索表格中输入需要查询的短语,然后提交即可。对于一个已经提交的检索,服务器方启动一个 CGI 程序进行响应。该程序对用户提交的各个表项进行合法性检查,通过后形成一个结构数据传给查询代理。

对于功能 2),主要用到动态网页生成技术和动态摘要算法。按照查询代理检索回来的数据,CGI 程序根据决定输出风格的模版,将结果数据整合到模版中,形成结果页面,输出给用户。动态摘要算法如图 5-3 所示,在文件 TSESearch.cpp 中实现。用户查询  $q$  的向量表示为  $q = \{t_1, t_2, \dots, t_m\}$ 。包含查询  $q$  的网页正文 Cnt,并且 Cnt 已经过滤掉网页标签等格式信息。为了保证用户检索的所有  $t_i$  都被加亮显示,摘要算法是在一个循环中实现的。其中 b2.1 是很必要的一步。以中文编码的网页为例,

每个汉字占用 2 字节空间,如果摘要算法不能够正确地判断一个汉字的起始位置,而是从汉字的一半位置开始,就会形成第三章图 3-2 中的部分摘要信息乱码。TSE 通过定位完整汉字的起始位置,可以正确显示摘要信息,没有乱码。

```

a. 初始化,保留 Cnt 开始字节数 resCnt = 48,每个查询词前后各保留字节数 resTerm = 256
   摘要内容 AbsCnt = Cnt 开始的 resCnt 字节
b. for each  $t_i$  in  $q$  do
   begin
     b1. 在 Cnt 中定位到  $t_i$  出现的位置 idx
     b2.1 如果  $idx > resNum$ ,从  $idx - resNum$  处到  $idx$  之间处找到一个完整汉字的
           位置  $cur\_idx$ ,  $AbsCnt +=$  从  $cur\_idx$  开始的  $2 * resNum$  字节。
     b2.2 否则,  $AbsCnt +=$  从  $idx$  开始的  $2 * resNum$  字节
     b2. 增加片断中间的分隔符,  $AbsCnt += "..."$ 
     b3. 加亮显示 AbsCnt 中的  $t_i$ 
   end

```

图 5-3 动态摘要算法

对于功能 3),查询代理接受用户输入的信息,记录到日志文件中。图 5-4 是 TSE 中的一段用户查询日志记录,包括用户查询时间和查询短语。

```

Wed-Apr-14-16:09:06-2004 北大网络实验室
Wed-Apr-14-16:11:00-2004 印度歌曲
Wed-Apr-14-16:15:10-2004 北大图书馆
Wed-Apr-14-16:15:23-2004 眼泪的故事
.....

```

图 5-4 用户查询日志的记录格式

## 2. 网页快照

搜索引擎索引的网页不一定是当前互联网上最新的网页,因此存在已经消失的可能性。为保证用户能够继续访问相应信息,搜索引擎一般都提供网页快照功能。

TSE 中的网页快照在第三章给出了图示 3-3,在文件 Snapshot.cpp 中实现。索引网页库是用第四章中第二节的方法生成的。

## 第三节 小 结

本章内容作为搜索引擎工作流程中的最后环节,负责把前两个阶段建立好的索引网页库、索引词表、倒排表提供给用户服务,这个交互的过程是通过查询代理完成

的。查询代理接受用户的查询请求,在倒排索引中查找符合要求的文档返回,并且提供网页快照功能。

本书上篇第三、第四和第五章这三个连续篇章,以设计并实现一个小的搜索引擎 TSE 为目标,讲述了目前流行的搜索引擎的基本特征,使读者可以快速对搜索引擎技术的整体有一个具体的认识,为进一步理解本书的中篇和下篇内容打下基础。

对搜索引擎技术感兴趣的读者,除了参考我们的 TSE 开放代码外(Tse 2004)(网址 <http://sewm.pku.edu.cn/src/TSE/>),还可以参考其他的开放代码的小搜索引擎程序,例如,Swish(网址 <http://swish-e.org>); htDig(网址 <http://sourceforge.net/projects/htdig/>); Clucene(网址 <http://sourceforge.net/projects/clucene/>)等。

## 中篇 对质量和性能的追求

上篇中,我们讨论了搜索引擎的基本工作原理,并通过一个实际的例子,阐释了这些原理在一个简单搜索引擎中实现的各个细节。同时,在上篇中我们也多次提到了性能问题和质量问题。尽管没有展开讨论,但其中的要素已经显示出来。事实上,这两个问题在搜索引擎的三个子系统中都有不同程度的体现。

中篇将围绕这些因素展开,具体来说,将讨论四方面的内容:

1) 一个并行搜集子系统的详细设计方案。尽管一个搜集子系统的硬件并行度不需要很高,但不做并行是达不到性能要求的。而能否在较短的时间内搜集到足够多的网页,也对搜索引擎系统后续的服务质量有直接影响。但只要是多个节点同时在网上搜集网页,就会引入新的技术问题。这将是第六章讨论的内容。

2) 网页净化与网页消重。在我们浏览网页,从中获取所需信息的同时,还会常常看见大量和我们所关心内容无关的“噪声”内容,如广告信息、版权信息等,有效地去除和网页主题内容无关的噪声内容,提取网页的元数据信息,如关键词、摘要、网页内容类别等,是 Web 信息处理的一项重要内容。在网页搜集的过程中,通过两个数据结构 `unvisited_table` 和 `visited_table`,我们可以完全避免对相同的 URL 执行多次网页抓取过程。但这并不保证抓到系统中来的网页都是不同的。Web 上大量的网页镜像和转载现象使得内容真正“独立的”网页要比实际搜集到的网页少很多。将相似的网页识别出来,当查询发生时只返回一个代表,这既是提高查询服务效率的需要,也是提高查询服务质量的需要。这是第七章的内容。

3) 高性能检索子系统。能够同时响应的用户查询数量,是商业搜索引擎的一个基本指标。与门户网站一样,商业搜索引擎也是以日访问量

作为基本的业绩指标,但和门户网站不同的是,每一个访问在搜索引擎上引发的过程要复杂许多。因此,设计一个能够承载每秒几十个,每天上百万人访问的查询子系统是一个非平凡的工作。

4) 相关排序与质量评估。从根本上讲,搜索引擎的服务质量体现在查询结果序列上。首先我们需要有足够多的结果来参加排序,这由搜集足够多的网页和良好的词典设计等技术来保证;然后是这些结果的排序问题。一般来讲,影响结果排序有两类因素,静态因素和动态因素。静态因素指的是和查询无关的因素,如网页内容的重要性、新颖性、权威性等,可以在预处理阶段形成;动态因素指的是和查询相关的因素,如和查询词的匹配程度,查询词在网页中出现的“分量”等<sup>①</sup>。它们是第九章讨论的内容。

---

<sup>①</sup> 例如,在同一篇网页中出现在<H1>和</H1>之间的词很可能要比出现在<H4>和</H4>之间的词更有分量。

## 第六章 可扩展搜集子系统

天网域名 e. pku. edu. cn, 寓意心有灵犀一点通, 就是鼠标一点(e.)天网就在用户的疑问与答案之间建起了通途。天网 1.0 采用集中式系统结构, 搜索量为百万级。为了扩大信息搜集规模, 让系统承载能力和中国网页规模增长的速度同步, 在 1.0 基础上推出了以中国全部 Web 信息为对象的天网 2.0, 为此我们重新设计了系统结构, 修改了实现方法, 设计了可扩展的 Web 信息搜集系统。本章将详细介绍此系统的设计和实现。

天网主要包括搜集子系统、索引子系统、检索子系统三个组成部分。可扩展的 Web 信息搜集子系统是天网核心部分之一, 由  $N$  个独立自主的集中式系统和协调模块组合而成。本章先介绍天网的总体系统结构, 再介绍集中式搜集系统的设计与实现, 接着重点介绍可扩展搜集系统。然后通过模拟实验与实际环境的实验, 分析说明可扩展分布式系统结构如何达到设计目标。

### 第一节 天网系统概述和集中式搜集系统结构

#### 一、天网系统结构

1997 年发布的天网 1.0 版采用单机搜集、单机服务的系统结构(我们习惯称之为集中式结构)不适应 Web 上信息规模的迅猛发展, 为此我们从 1999 年开始花了大约一年的时间设计和实现了天网 2.0 版的分布式并行系统结构。

系统分布的核心是数据的分布。对搜集部分而言, 实际是将 URL 分布在执行搜集任务的机器之间, 保证它们搜集的 URL 不会重复。对查询部分, 则是将索引数据分布在执行检索任务的机器之间。天网 2.0 系统概貌如图 6-1 所示。为了突出搜集和查询部分的并行化问题, 其中略去了搜索引擎三段工作流程中的预处理部分(尽管在大数据量时它也是需要并行的)。

搜集节点之间相互协调, 分配 URL, 保证一个 Web 主机的全部网页只能存在于一个搜集节点上。每个索引节点对应搜集节点搜集的网页, 查询代理节点通过多播向所有索引节点发送查询命令, 等待搜集到全部索引节点返回的检索结果后, 对所有结果依据相关度排序, 并缓存一定数量的结果, 最后向用户返回结果的首页。用户的后续查询(翻页), 将会在缓存命中, 不必再次启动后面的网络查询, 这将大大减少查询的响应时间, 降低后台查询系统的负载, 从而提高查询系统的性能。

第二章的图 2-3 示出了天网的详细系统结构。可以看出, 从功能模块上划分, 天



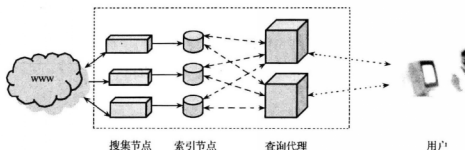


图 6-1 天网系统概貌

网系统由搜集子系统、索引子系统、检索子系统和日志挖掘子系统四个子系统构成。搜集子系统包括主控、搜集器和原始数据库；索引子系统包括索引器和索引数据库；检索子系统包括检索器和用户接口。日志挖掘子系统包括用户行为日志数据库和日志分析器。主控除了按照启发式算法优先选择重要的 URL 并分派给各个机器人外，还完成站点过滤、实现 robot 协议和域名解析并缓存等功能。搜集器按照 HTTP 协议负责从 Web 上抓取网页，为提高网页搜集速度，通常可以启动上百个搜集器同时工作。搜集器同时对搜集回来的网页内容进行分析处理，包括调用切词软件以提取关键词和摘要，提取 URL 超链，记录网页的元信息（如作者、修改日期、长度等），并将这些内容存入原始数据库。索引器将原始数据库的内容重新组织，建立索引数据库，以提高检索效率。用户接口在截取用户的查询请求后，将它转发给检索器，检索器根据查询项和索引数据库的内容，找到匹配的网页后，进行相关度计算并排序，然后通过用户接口返回给用户。另外，用户接口程序还将用户行为信息（包括用户查询项、用户点击的 URL、用户翻页情况等）记录到日志数据库。日志分析器用于跟踪用户行为，以提高搜索引擎的服务质量，如可以学习新词来动态更新词典内容。

## 二、集中式搜集系统

搜集系统包括主控模块、搜集器和原始数据库。主控模块是其中的控制模块，它主要负责：

- 1) 与网页抓取和分析进程的交互：发送配置信息，发送 URL，接收分析结果。
- 2) 与原始数据库的数据交互。
- 3) 访问控制：智能导向，robots 协议，主机访问频度，IP 地址等的控制。
- 4) 与外部系统的接口。

### 1. 系统设计目标

- 1) 主控与网页抓取和分析进程的分布。在系统设计中，必须采用分布式技术将

任务分布到多台机器上并行的处理。海量网页独立的分布在网络上,对并行访问提供了充分的可能性和合理性。同时,分布并行还会节省网络带宽资源。

2) 可定制性。系统可以让用户依据自己对信息的兴趣,配置用于引导系统搜集的导向词,以及搜集的范围。

3) 良好的开放性。尽可能使用和遵循现有的标准和协议,加强与其他系统交换信息的能力,包括支持 HARVEST 系统的 SOIF 信息格式和信息内容,以及 robot 协议。

4) 一定的扩展性。系统能在 CERNET 的网络环境下有效地运行,不需改动或改动很少就能适应不同的需要。

5) 一定的智能化。为了提高整个系统的查全率、查准率及搜索速度,搜索的内部算法应具备一定的智能化。

## 2. 系统结构和主要设计思想

集中式系统结构主要体现在主控的系统结构上。主控系统结构如图 6-2 所示。可以看出主控系统由 6 个进程组成,这些进程的功能如下。

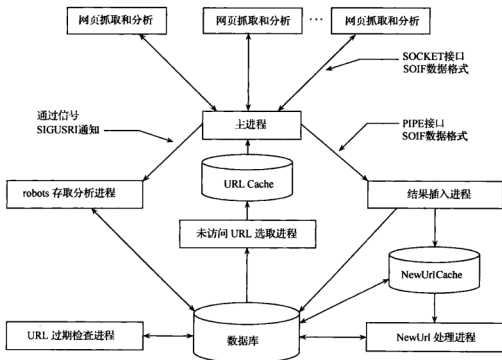


图 6-2 搜集系统的主控结构

“主进程”负责产生其他 5 个进程,接受“抓取分析进程”的连接并交互。

“结果插入进程”通过 PIPE 接收主进程得到的访问结果,通过各种检查后存入

数据库已访问网页列表。其中新的 URL 放入 NewUrlCache 或者新的未处理 URL 列表(当 NewUrlCache 满时)中,等待“新 URL 处理进程”进行处理。

“robots 存取分析进程”得到 URL 选取进程发出的 SIGUSR1 信号后,检查主机表中需要处理的表项进行存取分析。

“URL 过期检查进程”定期检查数据库中已经访问的网页,如过时,则将该主页的 URL 放入未访问 URL 列表中。

“未访问 URL 选取进程”从未访问 URL 列表选取合格的未访问的 URL 放入缓存中。

“新 URL 处理进程”从 NewUrlCache 或新的未处理 URL 列表中,取出新 URL 进行处理。

主进程与网页抓取和分析进程之间的交互是通过 SOCKET 来实现的,这样主控与存取分析进程可以运行于不同的机器上。它们之间是通过 SOIF 数据格式进行交互的。另外,敏感内容监控程序为了实时监控,也向主控发 GETS 的 SOIF 数据包,主控对此连接加以标识,每当出现敏感内容时就通过此连接发送 PUTS 的 SOIF 数据包。系统所用的 SOIF 数据包如表 6-1 所示。

表 6-1 SOIF 数据描述

数据包格式	描 述
@GETO{- }	存取分析进程发向主控,索取导向词
@PUTO{- Orientation-words{n}; Sensitive-words{n}; }	主控响应存取分析进程,发回导向词
@GETU{- }	网页抓取和分析进程发向主控,索取要进行存取分析的 URL
@PUTU{URL Last-Modification-Time{n}; }	主控发向存取分析进程,响应 GETU,发送 URL
@PUTR{URL Result{n};→result code [ Abstract Author Description File-Size	存取分析进程向主控返回结果 ——结果码 ——以下项可能没有

续表

数据包格式	描 述
Keywords Last-Modification-Time Title Type URL-References Time-to-live Weight Code-Type Retry-After Chars ] }	    ——含有 anchor 的权值, 范围 1~1000  ——此文章的权值, 范围 1~1000 ——此文章的编码  ——文章的权值较高的单个中文字
@GETS{- }	敏感内容监控进程发向主控, 表明自己实时地需要敏感信息
@PUTS{URL Sensitive-number{n}; }	当主控发现敏感信息时通过此数据包将信息发送给监控进程。 URL 为敏感内容的地址, Sensitive-number 为目前为止发现的敏感文章的总数

主进程与结果插入进程之间是通过 PIPE 进行通信的, 当主进程收到 PUTR 的数据包时便转发给结果插入进程, 由结果插入进程处理结果。插入进程对新的 URL 不作处理, 而是放入新 URL 的缓冲区, 或当缓冲区满时放入数据库。

新的 URL 最终由新 URL 处理进程统一处理。重复的抛弃, 未访问的插入数据库的未访问表中。

主进程与 robots 存取分析进程的通信: 当主进程将新的主机插入到数据库中的主机表后, 就向 robots 存取分析进程发 SIGUSR1 信号进行通知。robots 存取分析进程在信号处理过程中置标记, 随后在主机表中查询到相应的表项进行处理。

主进程与 URL 选取进程的通信是通过共享内存来进行的。将共享内存组织成循环队列, 主进程与选取进程通过 PV 操作来实现互斥区的操作。

可以看出, 通过这种设计, 系统的功能及模块的划分比较清晰, 主控模块中多个进程并发工作极大地提高了搜集处理信息的速度。另外, 系统在选取未访问 URL 时和处理新 URL 时使用了两个缓冲, 未访问 URL 选取进程不必每次等待主进程取走选取结果而可以继续选取。结果插入进程也不必等所有新 URL 处理完才从主进程接收下一个结果进行处理。这样, 进一步提高了进程间的并行程度。

管理员可以根据实际情况, 将主控和网页抓取和分析进程进行分布和组合, 也可以动态控制网页抓取和分析进程的数目。以达到分担服务器负载, 提高并发度, 加快信息搜集的目的。

在进行信息访问时,我们同时要遵循有关“Web robot”的各种约定:例如,不要在短时间内多次访问同一个服务器;获取“robots.txt”文件,不访问由它指定的目录等。为此我们在数据库中建立了一个主机表和一个禁止访问目录表,主机表记录了最近一次访问一个主机的时间,以及“robots.txt”文件的访问信息(没有、超时或成功访问)。只有在当前时间与此主机上次访问时间之差大于规定的时间间隔,并且不在禁止目录表中时,才允许访问此主机上的 URL。

每个节点网页抓取和分析进程与主进程之间信息交换的格式采用了扩展的 SOIF 格式,如表 6-2 所示。

表 6-2 SOIF 具体语法

SOIF	OBJECT SOIF   OBJECT
OBJECT	@TEMPLATE-TYPE{URL ATTRIBUTE-LIST}
ATTRIBUTE-LIST	ATTRIBUTE ATTRIBUTE-LIST   ATTRIBUTE
ATTRIBUTE	IDENTIFIER{VALUE-SIZE}DELIMITER VALUE
TEMPLATE-TYPE	Alpha-Numeric-String
IDENTIFIER	Alpha-Numeric-String
VALUE	Arbitrary-Data
VALUE-SIZE	Number
DELIMITER	;<tab>

从 SOIF 的语法定义中我们可以看出,每一个对象“OBJECT”包含一个模板类型“TEMPLATE-TYPE”,一个 URL 和一连串由字节数控制长度的属性和属性值的列表。由于是由长度分隔属性值,SOIF 属性值也可以是二进制的。SOIF 的格式易于分析,易于扩展,格式简单并且有足够的表达摘要对象的能力。它可以由软件自动生成,也可以手工建立。SOIF 有几个基本的属性,它们是:Abstract, Author, Description, Keyword, Title。

下面给出一个 SOIF 的例子:

```
@FILE{http://net.pku.edu.cn
Abstract{57}; HomePage of Computer Networks Lab of CS Peking University
Author{11}; Li Xiaoming
Title{12}; Networks Lab
Keywords{21}; networks computer lab
Type{4}; HTML
File-Size{4}; 1248
}
```

在摘要信息表示中,常用的属性有:Abstract, Author, Description, File-Size, Full-Text, Keywords, Last-Modification-Time, Refresh-Rate, Time-to-Live, Title,

Type, Update-Time, URL-References 等。

可以看出, SOIF 提供了通用的信息表示格式。不同的信息系统之间可以通过 SOIF 格式来交换信息, 从而达到共享信息的目的。我们的系统支持所有的常用属性, 可以通过 SOIF 与其他的系统交互, 具有良好的开放性。

目前 Web 上信息量庞大, 试图穷尽搜索比较困难。但是, 完全有可能在搜集的过程中尽量优先搜集用户配置的感兴趣的信息, 或者重要性较高的信息。这样尽管没有穷尽 Web 上的信息, 但是已经搜集的信息对用户的利用价值也会很高。这一点是通过加权的启发式搜索算法来解决的。每个 URL 有自己的权值, 反映了其文档内容的重要性, 没有访问的 URL 有一个预测权值, 访问后计算出其真正权值。访问时, 就依据权值的大小来决定优先顺序。我们是通过以下几个参数来预测 URL 的权值的: 父 URL 的权值, 即本 URL 所在文档的权值; URL 的 Anchor 权值, 这是出现在 HTML 文件中对 URL 的描述信息的权值; URL 目录长度权值, 每个 URL 都有一个目录(位于主机名和端口号后), 一般来说, 目录短的 URL 重要性相对高些; 被别的文档引用的次数; 加入者的信息; URL 的域名的深度。另外, 还有别的因素也会影响权值的计算, 例如, 如果用户定制的是优先搜集中文信息时, 中文编码的 URL 权值就应高些。

用户也可以为系统配置导向词, 当文档内容同用户配置的导向词相关度较大时, 文档的权值就会较高。在预测权值时, URL 所在文档和此 URL Anchor 信息与导向词的相关程度会影响预测权值的高低。

另外, 在 URL 发现过程中, 还可以主动的预测可能有价值的 URL, 例如, 得到了“<http://www.pku.edu.cn/a>”可以推测“<http://www.pku.edu.cn>”是一个有价值的 URL。这样做进一步增加了信息发现的主动性。

在 URL 被加入数据库时, 结果插入进程要调用过滤函数对 URL 进行过滤, 合格的 URL 才会被插入到数据库中。对 IP 地址的过滤, 主控提供了一个配置文件 field.conf。用户可以利用此文件定义 IP 地址的范围。

过滤重复的 URL 也是很重要的, 这样可以避免重复的访问和冗余信息。查重的时候, 首先要规整 URL, 排除“.”或者“..”或者 URL 中的编码, 例如, “foo/bar/./baz.html”与“foo/baz.html”是等价的。其次, 还要注意 URL 的主机名部分可能是 IP 也可能是此 IP 地址的多个域名。

对一个过期 URL, 进行重复访问更新时, 我们可以利用 HTTP 中的条件取操作, 即“*If-Modified-Since*”选项, 如果文档没有改动则系统不需要重新获取该文档。需要注意的是, 动态网页都不包括网页最后修改的时间, 不能用这种方法进行更新检查。

另外, 在系统进行信息发现过程中, 可能对域名解析的需求比较大, 往往有时在短时间内多次对一两个域名进行解析。这时, 可将域名解析的结果存入本地缓存, 不但会加快信息搜索的速度, 而且会减少解析域名带来的网络负载。

## 第二节 利用并行处理技术高效搜集网页的一种方案

Web 上成千上万的 WWW 服务器通过网页之间的链接构成的海量信息,各个主机之间的联系或多或少,但都可以说是相对独立的。单处理机系统受限于 CPU 的处理能力、磁盘存储的容量,不可能具备处理这种海量信息的能力,更不必说跟上 Web 信息的飞速增长了。

采用并行处理技术成为一个自然的选择。高性能并行计算机系统的种类有很多:SMP、NUMA、MPP、机群。比较起来,后者对我们的应用是最适合的。这不仅是由于其价格较低,还由于我们网页收集应用的基本特征:操作单纯,进程之间的通信量不大,对磁盘容量和聚集 I/O 吞吐率要求高,需要有经济效能较高的系统可扩展性以适应未来信息量增长的要求。这些都使我们认为机群是最好的选择。

在网页搜集子系统中,最基本的任务是一篇网页的抓取,它大致上要完成如下功能:从任务池中取一个网页的地址 URL,通过 DNS 得到其 IP 地址,用该 IP 地址与 Web 服务器建立 TCP/IP 连接,发 HTTP 请求,等待接收 HTTP 应答,关闭 TCP/IP 连接,分析收到的网页,将其中包含的新链接加入任务池中,将网页存放到磁盘数据库中。

由于网页抓取任务的这些阶段内容在完成时间上的不均匀性和不确定性(如取决于网络状态等),系统的并行性可以很自然地体现在两个层次:一个节点内部多个进程的并行和节点之间的并行。我们看到,如上所述系统微观上表现为明显的任务并行特征,但为了减少通信量,设计中考虑了一种在节点间采用数据并行、节点内采用任务并行的基本策略。即首先是确定一个在节点间动态划分网页 URL 的算法,保证不同的节点收集的网页不会重复,而在节点内由抓取进程自由获取下一个任务。

作为一个并行系统的设计,我们追求如下目标:单台机器的搜集能力不应随着搜集机器数量的增加下降很多,即要在追求负载均衡的同时将系统的通信和管理开销降到最低。同时,从实际运行出发,我们还要考虑系统的动态配置问题,即要允许在运行过程中添加和删除节点机器(这是因为一次搜集时间比较长,在这期间有时难免出现机器故障等,需要修复然后再投入服务;而在修复期间系统应该继续运行)。下面的讨论将按照如下要点展开。

- 1) 在节点间划分 URL 的策略;
- 2) 关于性能的讨论,包括负载均衡、通信开销和可扩展性等;
- 3) 性能测试和评价;
- 4) 系统的动态可配置性设计。

### 一、节点间 URL 的划分策略

为方便讨论起见,本节约定如下符号和术语。

$URLs = \{URL_1, URL_2, \dots\}$ , 所要完成收集的网页地址集合。这是一个开放和动态变化的集合。所谓“开放”, 指其中元素的个数是事先未知的, 具体有哪些元素当然也事先未知。在本文讨论的意义下,  $URLs$  的大小至少在千万量级。所谓“动态变化”, 指它在收集过程中随着新发现的地址增加。通常, 一次搜集过程由某些“种子”网页开始, 沿着它们包含的超链, 按照某种搜索策略(先宽, 先深, 等等)往下进行, 直到没有新的地址发现, 或者人为决定不要再进行了(如磁盘已满)。

$HOST(URL)$ , 一个网页地址的域名部分(或称主机部分), 通常对应某台 Web 服务器, 例如,  $URL = \text{http://net.pku.edu.cn/pp\_outline.htm}$ ,  $HOST(URL) = \text{net.pku.edu.cn}$ 。于是我们可以看到在  $URL$  上有一个按照域名的自然划分(即  $URL_1$  和  $URL_2$  同属这个划分的一个“块”, 当且仅当  $HOST(URL_1) = HOST(URL_2)$ ), 记为  $HOST(URLs)$ , 即它的每一个元素是一台主机上所有网页的集合。在不至于引起混淆的情况下, 我们也可以方便地将  $HOST(URLs)$  的元素看成是所涉及主机的域名。在本文讨论中国网页的意义下,  $HOST(URLs)$  的大小在 10 万量级(同样在不至于引起混淆的情况下, 我们有时也用  $HOST(URLs)$  直接表示它的大小, 于是符号  $HOST(URLs)$  的含义在本文是“重载的”, 由上下文确定具体哪一种)。

抓取进程, 负责根据一个  $URL$  完成一篇网页的抓取、分析和存储。它不负责更新任务池, 只是将得到的链接返回给下述协调进程。

让  $n$  表示并行收集系统的节点数,  $[n] = \{0, 1, 2, \dots, n-1\}$  表示网页搜集节点的集合。经验告诉我们每个节点启动 200 个左右并发抓取进程是比较合适的, 它们共享(争用)CPU、网卡和硬盘资源。从原理上讲, 我们可以考虑从  $URLs$  到  $200 \times n$  个进程之间的映射。但我们认为所带来的管理复杂性要比可能带来的好处大得多。于是我们考虑粗一些的粒度, 将目标设定在  $URLs$  上形成一个“优化的” $n$ -划分。而在节点内部的并发进程则以一种自由的方式来共同完成分到该节点的任务。

何为优化? 就是要在负载均衡和降低开销之间求得一种最好的折中。由于每个任务的执行时间可能相差很大(受网页的大小、网络状态的波动等的影响), 简单地按照任务个数的平均分配即使对于负载均衡也是没有意义的。

我们所采用方法的基本思路是对  $HOST(URLs)$  进行随机分配, 即建立一个从  $HOST(URLs)$  到  $[n]$  之间的映射。一旦一个  $HOST(URL)$  映射到了某一搜集节点, 该搜集节点就要负责  $HOST(URL)$  下面所有网页的收集。尽管不同的 Web 站点含有的网页数量会相差很大, 但由于  $HOST(URLs) \gg n$ , 并且如果我们的分配函数足够随机, 则可以认为分到各个节点上的网页个数比较均匀的可能性很大。

具体来说, 每个节点维护两张表,  $visited\_table$ ,  $unvisited\_table$ , 分别代表到目前为止本节点已经抓过的网页和需要抓取的网页。宏观上看, 整个系统有向量  $visited\_table[n]$ ,  $unvisited\_table[n]$ 。每个节点除启动若干抓取进程外, 还运行一个协调进程,  $controller$ 。协调进程的工作是根据本节点抓取进程和从其他节点的协调进程来



的信息,维护本节点的 `visited_table` 和 `unvisited_table`。下面是协调进程的工作算法如图 6-3,算法中从抓取回来的网页中解析出的超链接集合 `links` 包含的每一个 `link` 就是一个 URL。

```
for(i;)  
begin  
  a. 等待从其他节点传来的一个 URL,或者它所管辖的抓取进程返回的一个 URL 及相应网页。  
  b. 若得到其他节点传来的一个 URL  
      b.1 看 URL 是否已经出现在 visited_table 中,若没有,则将 URL 放到 unvisited_table 中;  
  c. 若得到从抓取进程返回的 URL,则从 URL 对应的网页中解析出超链接 links。  
      c.1 从 unvisited_table 中分给该抓取进程一个新的 URL,并将返回 URL 放到 visited_table 中  
      c.2 并对每一个超链接符号串 HOST(link) 进行模 n 数取,得到某个整数 i;  
      c.3 对每一个超链接 link 及其对应整数 i  
          c.3.1 如果本节点编号为 i,执行 b.1 的动作  
          c.3.2 否则,将 link 发给节点 i  
end
```

图 6-3 协调进程工作算法

从它的工作内容来看,每个节点的 `visited_table` 和 `unvisited_table` 是此消彼长的。`unvisited_table` 就相当于一个“任务池”,在整个收集过程中它先是逐渐变大,然后逐渐变小,理论上讲,是可以最终为空的。

我们还看到,在不同节点上运行的协调程序是需要通信的:发送不该本节点负责的 URL,接收该本节点负责的 URL。而从理论上讲(只要不同的 URL 对应不同的网页),这种两级并行的策略能保证网页不被重复抓取。

然而,互联网上复杂的现象使得这个理论的前提不成立。不同的域名可以对应相同的 IP 地址,例如, `net.pku.edu.cn` 和 `net.cs.pku.edu.cn` 都对应 162.105.203.25,从而沿着这两个不同域名的访问是等效的(这只需要在相应的 DNS 服务器上做适当映射就能实现)。这就意味着两个不同的 URL 可能对应相同的网页,于是该网页就可能被多次抓取。一种考虑是当收到一个 URL 时,首先得到 `HOST(URL)` 的 IP 地址,然后依据这个 IP 地址来发出 HTTP 请求。但我们看到,在某些情况下在 HTTP 请求中用 IP 地址(而不是用域名)会得不到所希望的 HTTP 应答。典型的情况就是大型门户网站为动态分配访问负载所采用的组技术:一个组的代表可以是某个“内部域名”,例如, `pagegrpl.sohu.com.cn`,但它可以包含若干 IP 地址,这些 IP 也可能根据应用的需要分成几个子组,例如,一个子组对应来自对 `www.sohu.com.cn` 的访问,另一个子组对应来自对 `news.sohu.com.cn` 的访问。当用域名引起的访问发生时, `pagegrpl` 服务器动态确定某个 IP 来提供服务,而不允许直接通过 `pagegrpl` 内的任何 IP 来进行访问。于是,就出现了这样矛盾的要求:为了不重复抓取同一个网页,需要在 HTTP 请求中用 IP 地址;但为了能够正常访问采用组技术的大型网站,在 HTTP 请求中不能用 IP 地址。我们目前的做法是照顾后者,即容忍了有些网

页的重复抓取(然后通过一个“消重”过程去除冗余)<sup>①</sup>。

这样,我们就得到如图 6-4 所示的一个系统运行示意图。其中的协调进程之间两两建立起连接,形成一个逻辑全互联关系,直接传递它们之间的交叉 URL。

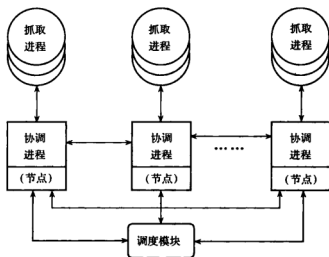


图 6-4 分布式 Web 搜集系统结构

在图 6-4 中调度模块(记为 WSR)有特别的意义,它维护系统内所有登记协调进程的信息,包括它们的 IP 地址和端口号。当任何一个协调进程的信息有所改变时,WSR 负责将更新的信息转送给其他协调进程,便于建立连接和变更连接。协调进程从 0 开始编号,直到  $n-1$ ,各自负责收集存储属于自己范围内的网页。每个节点上运行若干抓取进程,它们在协调进程的管理下工作。抓取进程负责接收从所属协调进程发送的 URL,抓取该 URL 指向的网页并传送给所属协调进程。各协调进程之间都建立有双向连接,可以全双工的工作。当任一协调进程发现自己的收集模块发回的网页中包含不属于自己的 URL 时,就将此 URL 传送给负责它的协调进程去处理。为减少通信量,各协调进程之间只传送 URL。

## 二、关于性能的讨论

从实际应用的不同需求出发,海量网页搜集系统的性能可以有不同的定义,涉及四个主要参数,完成一批网页搜集所花的时间( $T$ ,小时),收到的网页数量( $P$ ),系统和 Internet 之间的带宽( $B$ ,Mbit/s),参与搜集的机器节点数( $n$ )。如果按照平均每篇网页 15KB 数据量计算,注意到上述参数的单位<sup>②</sup>,我们有最基本的

① 还有比较复杂的情况是“不同域名,相同 IP,不同网页”,如 net.pku.edu.cn 和 i3s.pku.edu.cn。

② 1 字节等于 8 位,1 小时等于 3600 秒,带宽用 Mbit/s 为单位等。

$$T \geq \frac{8 \times 15 \times 10^3 P}{B \times 3600} \times 10^{-6} = \frac{P}{3B} \times 10^{-4}$$

例如,  $P=10^8$ ,  $B=100$ ,  $T \geq 33.3$  小时。当然,这是最理想的情况了。通常,如果  $B$  表示网络连接的额定带宽,  $\hat{B}$  表示实际达到的有效带宽,  $B \gg \hat{B}$ 。虽然有效带宽是随时间变化的,但在固定的环境下,同样一段长时间(如一天)的平均有效带宽基本上是稳定的,因此讨论  $\hat{B}$  有意义。下面是关于性能的几种考虑:

1) 在给定硬件条件(节点数,网络有效带宽)下,给定时间内搜集不同网页的数量。上面提到过,在一段时间的平均有效带宽是有意义的。一次搜集过程通常要历经几天甚至几周,因此用有效带宽来比较系统设计是合理的。

2) 给定硬件条件(节点数,网络有效带宽),达到某给定网页搜集量所花的时间(越少越好)。网页搜集过程告诉我们,刚开始会比较快,unvisited\_list 迅速增长,每一个抓回来的网页都带回来一些新的 URL。但随着过程的进行,进展会越来越慢,不仅新发现的 URL 少了,更重要的是新发现的 HOST(URL)少了,这导致搜集网页逐渐集中到几个大网站上,网站本身的吞吐能力限制了搜集速度,甚至引起比较多的 HTTP 应答失败。同时,由于网站的数量变少,负载均衡可能开始出现问题(HOST(URLs)  $\gg n$  不再成立)。这表现为随着搜集过程的进行,每天收到的网页越来越少。在实际中,人们可能先根据经验确立一个目标网页数量,达到后就停止。例如,我们在 2003 年初估计中国的网页数量在 1 亿以上,但超过 2 亿的可能性不大,再考虑到以先宽方式搜集时得到的网页的重要性随时间快速递减(Najork et al. 2001),于是搜集 1 亿左右就停止会是一种合理的考虑。

3) 在给定时间( $T$ )内,完成搜集给定目标网页数量( $P$ )所需的节点个数( $n$ )。有许多因素使得给定  $P$ ,  $n$  并不和  $T$  成线性反比。较小的  $n$  意味着许多好处:较小的节点之间通信开销,较少的外部通信资源(出口带宽)冲突,较好的负载均衡。因此,如果我们有一个搜集时间长度(如两周),确定一个目标网页数量(如 1 亿),也许用  $n=20$  会提前 3、5 天,但用  $n=5$  可能也能满足要求。

无论哪一种标准,性能的瓶颈可能在不同的条件下表现在系统的不同部分。如前所述,抓取一个网页的任务包含有许多阶段,其中涉及系统的不同部件:处理器,磁盘,网络带宽。例如,当节点数比较少时,处理器和磁盘会是个瓶颈,因为每个节点需要启动较多的抓取进程,否则达不到所需的抓取能力。当节点较多时(如大于 10),网络带宽就成为主要矛盾。Internet Archive 的 Brewster 告诉我们,他用 6 台机器, T1 连接,每次抓两个月,能得到  $2 \times 10^9$  左右网页, 30TB 数据量。 $30\text{TB}/60 = 0.5\text{TB}/\text{天}$ ,需要 46Mbit/s 以上的平均有效带宽。<sup>①</sup>

<sup>①</sup> 在本书即将脱稿之际,谢正茂为天网设计的一个新型搜集系统在千兆带宽上也达到了每天搜集 5000 万网页(超过 0.5TB 数据量)的水平。

### 三、性能测试和评价

上述系统设计首先通过一个类似于“trace driven simulation”的方法进行了模拟。具体做法是在一个单节点网页搜集系统的正常运行过程中加入数据采集程序，产生并行算法需要使用的模拟数据。对于每个网页，采集了它的 URL 和所包含的 URL 链接，总共得到了 761129 篇网页的信息，数据量大小为 507MB。以此作为我们并行算法模拟的输入，分别考察了节点数  $n$  为 2、4、8、16 四种情况。为对比起见，在每次运行多节点模拟时也同时运行单节点的模拟。下面是从几个方面对实验结果的评价。

#### 1. 负载平衡分析

如前所述，我们为系统负载平衡采用的基本技术是利用散列函数在节点之间动态分配 HOST(URL)。由于这里要考虑的是一个过程，不能仅仅用过程结束后每个节点总共分得了多少 URL 来评价是否平衡。因此，我们通过分析每个节点每小时搜集网页数来评估负载平衡的效果。具体来说，模拟程序以小时为数据采集的单位，考察了系统前 10 小时运行结果。

设系统中包含两个节点，在一个小时内节点一搜集的网页数为 2(即  $n=2$ )，节点二搜集的网页数为 3(由于我们只关心两行数据的差别，这里用一些小整数是无关紧要的)，依此比例进行下去，取前十小时作为一组参照序列。规整化后如表 6-3 所示，其中  $t$  表示时间(小时)， $i$  表示节点编号，中间的数据表示“节点  $i$  在  $t$  小时里得到的网页总数”。

表 6-3 参照序列,假设节点数为 2

$t \backslash i$	1	2	3	4	5	6	7	8	9	10
1	2	4	6	8	10	12	14	16	18	20
2	3	6	9	12	15	18	21	24	27	30

如果四组实验数据好于参照序列，认为达到负载平衡要求。规整化四组实验数据和参照序列数据，计算出的数据表示成图 6-5。在图 6-5 中三角形线表示参考数据方差，方形线表示节点数为 2 时的方差，钻石形线表示节点数为 4 时的方差，加号形线表示节点数为 8 时的方差，星号形线表示节点数为 16 时的方差。从图中可以看出任何一组实验数据的方差都小于参考方差。而且随着节点数的增加，平衡性越来越好。在节点数为 16 时，几乎和  $x$  轴重合。说明各个节点搜集的网页数基本相等，因此并行搜集子系统负载平衡达到预期目标。

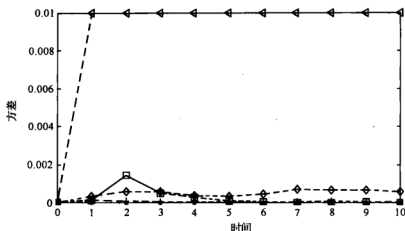


图 6-5 负载方差

## 2. 通信开销

实验中为保证环境一致性而采用集中解析域名方式，因此各个节点之间只有交叉 URL 的通信量。各节点之间传递的只是 URL，每个 URL 长度设定最大为 128 字节。在今后的实际系统中，为提高域名解析的利用率，各节点已经解析出的主机名要互相传送，每个节点都维护一份当前系统已解析出的主机名与 IP 地址的对应表。各个节点中的表要保持一致，这也构成主机通信的一部分。每个主机名与 IP 对应关系的存储长度不会超过 72 字节（用 64 字节存储主机名，4 字节存储 IP，4 字节存储访问时间），因此节点之间通信量不大。另外，为了实现系统的动态调度，需要在增减节点时，将现存节点中维护的相应表进行修改，为了保证表的一致性，需要在各个节点之间互相传递信息。这种情况并不会经常出现，因此不会过多的增加节点通信量。考虑在上述后两种情况下的节点通信中，一个副本要传送给多个节点，在实际系统的运行中我们采用组播技术。

## 3. 可扩展性

图 6-6 是模拟系统设计可扩展性的具体结果，考察了系统节点数  $n = 1, 2, 4, 8, 16$  等情形。其中，横轴表示目标搜集系统的运行时间，单位为小时；纵轴为累积搜集的网页数。图中分别显示了在 5 种不同的系统规模下前 10 小时的运行结果。

模拟过程是在一台机器上，通过多个进程的协调进行的，体现了本章第二节描述的方案和算法，以及如图 6-4 所示的系统模型。

将图 6-6 的内容综合起来，我们得到图 6-7。X 轴代表节点数，Y 轴表示  $n$  个节点协同工作搜集的网页数与单个节点在同样时间段里搜集网页数的比（称为加速比）。由图中可以看出加速比随着节点数的增加基本上是线性增加，因此并行系统具

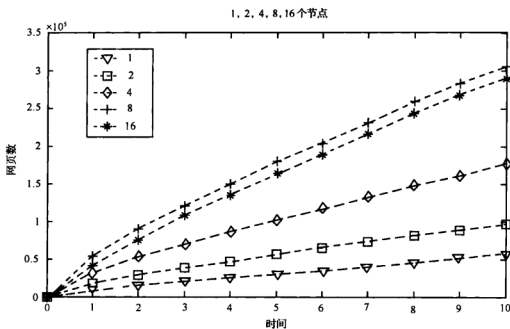


图 6-6 并行搜集系统与集中式搜集系统的性能对比

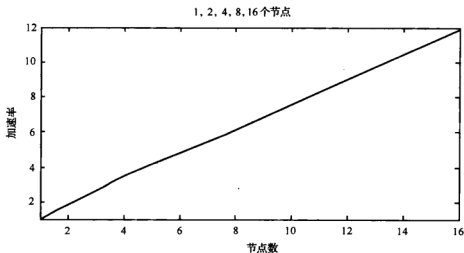


图 6-7 分布式系统效率

有良好的可扩展性。关于这里介绍的并行搜集子系统更多的细节见参考文献(Yan et al. 2001a)。

#### 四、系统的动态可配置性设计

并行系统中 WSR 模块的存在,使每个节点都能够保持当前系统中所有节点的最新信息,是系统动态可配置性的前提。在保证系统负载均衡的条件下,我们考虑三种方法保证系统具有动态调度性:

1) 采用散列函数动态调度 URL。

2) 第二个方案是结合第一种方法,同时每个节点记录着一张 WWW 主机表,这张表在各个节点是相同的,其中每一条记录包含一个 WWW 主机及其所对应的一个节点。

3) 采用逻辑上二级映射的方法。首先用散列函数映射 URL 到一张逻辑表上,然后将这张表上的相应部分映射到各个节点。

对以上三种方法通过对节点数增减 1 的情况分析,可以知道动态调度性的好坏,设 WWW 主机数为  $m$ ,系统初始节点数为  $n$ ,  $n_i$ 、 $n_j$  表示系统中任意两个节点,加 1 情况记为  $n \rightarrow n+1$ ,减 1 情况记为  $n \rightarrow n-1$ 。

第一种方法,系统初始,每个节点负责的主机数为  $m/n$ ,散列函数对  $n$  取模,可以保证系统负载均衡。在  $n \rightarrow n+1$  情况下,散列函数对  $n+1$  取模,此时可以保证系统负载均衡,但是由于模数改变使原先分配给节点  $n_i$  负责的 URL 可能分配给  $n_j$  负责,从而导致重复搜集一些已经搜集过的网页;在  $n \rightarrow n-1$  情况下具有同样的缺陷。

第二种方法,为了克服第一种方法存在的缺陷,每个节点需要附加存储两个表:主机表和本节点已经访问过的 URL 表。每个节点保存的主机表相同,已经访问过的 URL 表各自不同。WWW 主机数目有限,后加进的节点没有足够的 WWW 主机去进行搜集,为达到负载均衡的要求,需要其他节点迁移一部分主机及其所附带的已访问过 URL 的信息给新加入的节点。此时 URL 经过散列函数处理后,需要额外采取一个步骤:在一个节点判断得到该 URL 属于自己负责后,应先根据主机表判断此 URL 对应主机是否已经被其他主机负责,如果没有,才应该自己处理;同理,在一个节点判断得到一个 URL 属于另一个节点(不妨设为 A)负责后,应先根据主机表判断此 URL 对应主机是否已经被其他主机负责,如果没有,才将该 URL 传给 A。这种方法由于需要维护各节点之间主机表一致,节点数目变化时需要传送主机和附属 URL 信息,较第一种方法增加了主机之间的通信量,我们决定将每个 URL 采用 MD5 算法,用 16 个字节表示每个 URL。

第三种方法,采用逻辑二级映射的方法。其中一级逻辑节点可以用数组形式存储(不妨设这个数组为 A),每一个数组元素的下标即是它所代表的逻辑节点号,其中存储该逻辑节点对应的节点序号。举例说明,设  $n=10$  为 10 个物理节点,  $m=50\,000$  为逻辑节点的数目,系统初始如图 6-8(a)所示:其中 A[1]到 A[50 000]称为逻辑节点。URL 经过散列函数处理后首先平均映射到逻辑节点上,再经过第二次映射将第 1 到第 5000 的逻辑节点映射到 1 节点,第 5001 到第 10 000 的逻辑节点映射

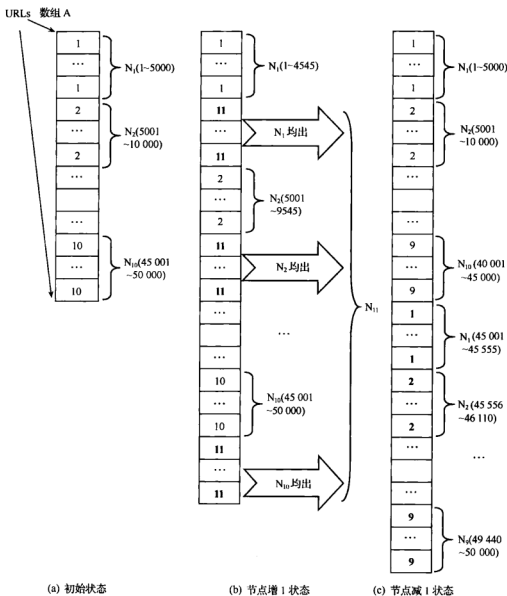


图 6-8 URL 两阶段映射

到 2 节点,依此类推将逻辑节点平均分配到 10 个节点上。

当节点数增 1,每个节点都要让出一部分控制的逻辑节点给新的节点,即对 A 作部分修改。如图 6-8 (b)所示, $n_{11}$ 是由  $n_1, n_2, \dots, n_{10}$  匀出部分组成,其逻辑数组中对应的项被置为 11。当节点数减 1,如图 6-8(c)所示,减掉的节点要让自己控制的主机,平均分配给其他节点,同样要修改逻辑数组中的值。

同第二种方法比较,这种方法多一次映射,同样每个节点需要保存自己访问过的



URL,随主机迁移时需要转给新的节点。但是这种方法可以不必存储主机表,每个节点中只要保存一份同样的数组 A 就可以了;这样就减少了节点之间的通信量,并且系统配置改变后,不必像第二种方法一样在散列函数处理后作附加的判断步骤。

现在,由于第一种方法实现简单,在模拟系统中我们采用这种方法。第三种方法较前两种方法优越,是我们系统实现的选择,以保证系统具有良好的可动态配置性。关于这部分介绍的系统动态可配置性的更多细节见参考文献(Yan et al. 2001b)。

### 第三节 天网分布式搜集系统

天网分布式搜集系统 P\_Arthur 由三个独立运行的模块组成:king, queen, mosquito。其体系结构如图 6-9 所示。

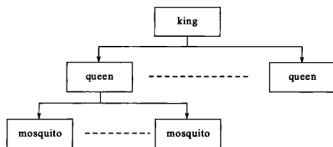


图 6-9 天网分布式搜集系统 P\_Arthur 体系结构

在本系统中,king/queen 属于 url 调度模块,mosquito 负责具体的搜集任务,mosquito 是最终执行任务的站点搜集器。

king:负责分配站点给指定的 queen 进行收集。queen 会定期的报告当前的运行情况,king 根据 queen 报告的数据,对收集任务进行按站点分配(site→queen),新出现的站点总是分配给当前负载轻的 queen。对系统而言,king 并非必需。

queen: 是核心的调度模块,主要功能有三个。

1) 分配搜集任务。queen 内部缓存了等待收集的站点列表,queen 分配一定数量的 mosquito 来收集该 queen 负责的站点任务。当一个 mosquito 完成收集后,会把当前状态报告给 queen,queen 回收该进程。queen 也会收集由 mosquito 发回的 URL 集合。

2) 与 king 进行通信,定期报告当前状态,接收和发送需要收集的 URL 集合。

3) 与其他的 queen 进行通信。queen 会从 king 下载 site→queen 的列表;对未收集的 URL,如果当前 queen 知道其负责收集的 queen,会直接把该 URL 转发给负责收集该 URL 的 queen,而不需要发给 king。

queen 其他的一些功能如下:

1) 加载 allow 列表。仅允许收集列中有的站点,allow 列表中可以用域名或者 IP 地址来指定。

2) 加载 deny 列表。不收集 deny 列表中的站点。对于不在 allow 列表中 IP 地址或域名,或在 deny 列表中的站点,搜集系统均不会收集。

mosquito:负责收集指定站点的网页,对不属于该站点的 URL,会发送给 queen。queen 负责把该 URL 发送给 king,queen 或自己保留。

P\_Arthur 搜集系统包括收集网页和合并网页数据两部分。具体的源程序和运行方法,请参看网址 <http://sewm.pku.edu.cn/src/paradise> 中的搜集(crawl)模块。

## 第四节 对 DeepWeb 的认识

Deep Web(又称为 Invisible Web 或 Hidden Web,暗网),理解其含义可从与之相对的概念 Surface Web 入手。Surface Web 指一般的被搜索引擎索引,用户可以检索到的网页。而 Deep Web 则是通过前述渠道无法获得的那部分网络资源,这些信息通常锁定在数据库内,可以响应具体的查询而生成网页。虽然这些动态的网页拥有唯一的 URL 地址,用此地址可以再次检索到,但它们不是稳定不变的,或者不是按静态页面存储的,相互之间也没有链接。Deep Web 还包括需要注册的或者对页面限制访问(例如,采用“拒绝 Robots 访问标准”(Robots Exclusion Standard)),禁止搜索引擎浏览和生成缓冲拷贝的网站。只有通过 JavaScript 和 Flash 生成的链接才能访问的网页也属于 Deep Web,因为搜索引擎无法正确识别出这些链接。据估算,在 Deep Web 上,信息量比传统搜索引擎“知道的”信息量大几倍到几个数量级(Bergman 2001)(Sherman et al. 2001)。

给出两个提供人才招聘的 Deep Web 网站的入口页面,有兴趣的读者可以试一试类似的网站,如图 6-10 所示。

值得注意的是,许多非文本文件,如多媒体文件,Usenet 存档和非 HTML 格式的 pdf、doc 等文件过去是 Deep Web 的一部分,但是现在大多数搜索引擎都能够索引这些资源。

### 一、Deep Web 的成因

Bergman 关于 Deep Web 的调查报告(Bergman 2001),得出如下结论:

- 1) Deep Web 包含的可访问公共信息容量是 Surface Web 的 400~500 倍。
- 2) Deep Web 包含 7500TB 的信息,而 Surface Web 包含的信息容量只有 19TB。
- 3) Deep Web 包含 5500 亿独立文档,相对应的 Surface Web 只包含 10 亿个。
- 4) 现有的 Deep Web 站点估计超过 100 000 个。
- 5) 60 个最大的 Deep Web 站点就已经包含 750TB 信息,超过 Surface Web 所包含信息的 40 倍。



图 6-10 人才招聘网站首页

6) Deep Web 站点的月访问量比 Surface Web 站点高出 50%，并且与 Surface Web 站点相比有更多的链接。可是那些典型的大型 Deep Web 站点在互联网搜索领域却不为众人所知。

7) Deep Web 是互联网新信息增长的最大来源。

8) Deep Web 站点包含的信息在内容范围上比一般 Surface Web 站点更深。

9) Deep Web 包含的有效高质量内容总量至少是 Surface Web 的 1000~2000 倍。

10) 超过一半的 Deep Web 内容都保存在专业领域的数据库中。

11) 95% 的 Deep Web 信息都是面向公共访问的,而不是需要付费或者订阅的。

Deep Web 资源内容更为丰富,专业性较强,质量比 Surface Web 好得多,但是如果用户不知道信息的确切位置,利用普通搜索引擎是无法找到的。Deep Web 是相对不可见,并非不可检索,而只是普通搜索引擎无法搜集索引。既有技术方面的原因,如网站管理者特意屏蔽,搜索引擎的搜集器遇到无法解析的数据格式;又有非技术原因,如知识产权保护的负面影响,搜索引擎成本限制等。

由于技术原因导致网页内容对于搜索引擎不可见, Sherman 和 Price (Sherman et al. 2001) 总结如下原因。

1) 未被链接的网页。因为搜集程序只能搜集链接到的网页和向搜索引擎提交链接的网页,所以新出现的网页如果未被链接,或者网页处于隔离状态,就不会出现在搜索引擎的查询结果中。

2) 文件格式。特殊的文件格式,如 Postscript, Flash, Shockwave, Microsoft Office 文档(包括 Word, PowerPoint, Excel 等),可执行程序,压缩文件(如 zip, tar)等。尽管技术上可以索引,但是大多数搜索引擎出于经济原因的考虑是不全索引的。

3) 网页文字内容信息过少。这类网页主要由图片、声音或视频组成,很少或者几乎没有文字信息被提供给搜索引擎来理解网页的意思。

4) 关系数据库文件。要访问数据库内的信息,必须正确填写一个或多个表格,因为搜集程序无法模拟人的行为自动填表格,所以无法完成交互过程获取数据库内的信息。

5) 实时内容的信息。包含股票报价、天气预报、航班信息等变化很快,同时信息的历史价值有限的网页,因其巨大的数据存储需求会加重服务器的负担,因此不在一般搜索引擎收录的范围之内。

6) 动态产生的网页。垃圾站点会制造搜集程序陷阱,搜集程序一旦陷入,会陷入死循环,影响搜索引擎的正常工作,因此搜索引擎都会限制行此类网页。

从非技术原因考察 Deep Web 的形成, Price 和 Sherman (Price et al. 2001) 总结如下原因。

1) 不透明网页(The Opaque Web)。不透明网页指搜索引擎可以索引但未索引的网页,主要由以下几个因素造成:①搜索深度。受经济因素制约,早期的搜索引擎基本上都只索引网站主页等表层网页,虽然随着 Spider 爬行成本的降低,其搜索深度也在不断增加,但还是会受到深度限定而无法爬行任何站点的任何网页,这样一来,一部分超过其搜索深度但有价值的网页就成为不透明网络。②搜索的最大数。即使在搜索深度之内, Spider 爬行的网页数也有可能超过其最大容量,这样 Spider 就不得不舍弃其中的一部分,这一部分也成为不透明网络。③搜索频率。互联网上的信息日新月异,而搜索引擎 Spider 爬行的速度有限,大概为每天千万量级,这样,在 Spider 再次光顾之前,新出现的网站(页)也就成为不透明网络。此外,链接中断

或未被链接的网页等也可能形成此类网络。

2) 私人网络(The Private Web)。这类网页本来是可以索引的,由于网页主人加了口令保护、禁止索引的网站标记等,便将搜索引擎的 Spider 拒之门外。

3) 专有网络(The Proprietary Web)。专有网络是指只对注册用户(免费注册用户和收费注册用户)开放的网页,这部分网页都需要用户输入用户名和密码,其资源才可被利用。机械式的搜索引擎无法填表注册,当然也就无法索引。

4) 真正看不见的网页(The Truly Invisible Web)。真正的隐藏网络主要包括非 HTML 格式的文档、动态网页、实时数据及网络数据库,其形成原因主要包括:①目前大部分搜索引擎只能索引 HTML 网页文档。HTML 文档都有“head”和“body”两部分,搜索引擎根据“head”来判断其为 HTML 文档并加以索引,而对于其他如 Postscript、PPT 和 DOC 等格式的网页,搜索引擎的开发商出于成本等多种考虑而不加以索引,这样,非 HTML 格式的网页内容都深藏在信息海洋的海底。不过随着检索服务的逐步完善,一部分搜索引擎可以索引多种格式的文档,如 Google 现在已经可以检索 Image 以及 PPT、PDF、DOC 和 PS 等格式的文档。②Spider Trap。Spider 一旦陷入恶意的 Spider Trap 的程序中就有可能陷入死循环而影响搜索引擎的正常工作,因此搜索引擎都会限制 Spider 爬行此类网页而使之成为“漏网之鱼”。③搜索引擎无法索引动态网页(Dynamical Page)及网络数据库。现在 Internet 上存在着海量的数据库资源,其中很大一部分资源可以免费获取,但都是以数据库为后台,动态网页技术和数据库技术相结合的方式组织资源,数据库根据用户输入的检索式返回符合检索要求的动态网页。而 Spider 都是沿着超链接漫游,根据超链接提取新的 URL,无法完成“输入检索式”的动作,因此也就无法对其加以索引。

以上从技术角度和非技术角度对 Deep Web 的成因进行总结,可以看到有的内容是交织在一起的,因为只有这样,才能全面概括 Deep Web 的成因。

## 二、搜索 Deep Web 的方法

### 1. 利用已有的相关检索工具

网络信息内容并非真正隐藏起来,只是有些一时难以获得。网络信息可见是绝对的,看不见是相对的。随着人们逐渐重视,越来越多的 Deep Web 资源可通过多种工具检索出来。Lackie 维护的 Deep Web 搜索方法汇总可以作为初次接触 Deep Web 并进行检索的入口(Lackie 2006)。

#### (1) 专业目录(Selective Directory)

Deep Web 专业目录也是一个主题指南,它按照一定规则排列大量网站的链接,用户通过专业目录可得到与某一主题相关的网站信息,然后点击链接。这类工具一般既提供分类导航服务,又提供关键词检索服务。例如,用户检索“searchengine”,出来的结果是研究“searchengine”的站点列表,而不是直接有关“searchengine”的资料。

这个大类下的检索工具为数不少,下面以 FindLaw 和 InfoMine 进行介绍。

1) FindLaw (<http://www.findlaw.com>)。FindLaw 是 Internet 上为律师、商人、学生、法律专家、公众、法律团体等提供法律资源的综合性法律网站。其特点是资源极其丰富,提供简单检索、高级检索、名称检索、法律网站检索、法律数据库检索等多种检索方式;FindLaw 的另一特点是能及时发布法律界的最新消息和提供法律专家案例分析,且其中的“*ForLegalProfessionals*”为用户提供了系列免费数据库和法律相关的主题信息。

2) InfoMine (<http://infomine.ucr.edu>)。InfoMine 是一个由加利福尼亚大学图书馆、加利福尼亚州立大学图书馆等机构建立的虚拟图书馆,主要用来为大学教师、学生 and 研究人员提供网络信息服务。InfoMine 提供数十万个学术站点的链接,内容包括数据库、电子期刊、电子图书、BBS、邮件列表、论文、在线图书目录等学术资源,为方便用户检索将站点分为商业与经济、政府信息、人文社会科学、生物、农业与医药等 9 大类。

3) CompletePlanet (<http://www.completeplanet.com>)。CompletePlanet 是 BrightPlanet 公司提供了可以检索 7 万多个 Deep Web 数据库和专业搜索引擎入口功能。

### (2) 主题明确的 Deep Web 数据库

主题明确的数据库 (Subject-specific Databases) 主要用来提供动态信息和数据库中资源的查找,可以弥补普通搜索引擎 Spider 无法爬进网络数据库的技术缺陷。目前比较优秀的有 AnimalSearch、Educator's Reference Desk 和 PubMed 等。

1) AnimalSearch (<http://animalsearch.net>)。提供动物的相关信息,可以按动物的类别、种群、地域等分类进行检索的数据库。

2) Educator's Reference Desk (<http://www.eduref.org>)。拥有 2000 多个由美国各州教师提供学习计划,用户可通过艺术、计算机科学、健康、数学等主题进行浏览;包含 3000 多个网络教育信息站点的 URL,按主题分为参考咨询、普通教育、教学、家庭生活等 12 大类,每类又细分若干小类;收录 AskERIC 1992~2003 年的 200 多个咨询文档,包括 ERIC 引文、Internet 站点、讨论组等,按教育管理、家庭生活、普通教育等主题组织;提供 ERIC 数据库的入口和 Gateway to Educational Materials (GEM) 的链接。

3) PubMed (<http://www.ncbi.nlm.nih.gov/sites/entrez>)。PubMed 是美国国家医学图书馆 (NLM) 下属的国家生物技术信息中心 (NCBI) 开发的。包含 1800 多万条生物医学文献摘要,每年增长 70 万条。其中 PubMed Central 已经收录超过 200 种期刊,50 万个全文文献。

### (3) 针对 Deep Web 的搜索引擎

用户使用搜索引擎进行检索,返回的是用户所需要的信息资源。因为搜索引擎能根据用户的检索要求,到各网络数据库进行检索并返回符合检索要求的资源,而不

同于选择性指南那样注重引导、浏览。目前已有 Deep Web 专业搜索引擎 (Specialized Search Engine), 如 Singingfish、Scirus、UFOseek 等, 现简单介绍多媒体搜索引擎 Singingfish。

1) Singingfish 曾经是全球最大的免费流媒体 (Streaming Media) 专业搜索引擎, 2007 年停止服务。主要特点: ①资源丰富、分类详细, 将所收集的多媒体信息按音乐、新闻、电影、电视、广播、体育、金融与其他等项目分类, 并允许用户选择检索文档的格式, 如 mp3、Real、QuickTime 以及 Windows 文件; ②Singingfish 允许 Web 上任何站点 (搜索引擎、目标站点或门户网站) 传递符合用户检索要求的多媒体信息如音频、视频等给 Singingfish 并返回给用户。

2) Scirus (<http://www.scirus.com>)。Scirus 是 Elsevier 开发的综合性科学知识搜索引擎, 提供 4 亿 4000 万多个具体的科学网页, 其高级检索是可供用户选择检索信息的类型。Scirus 在收集具体的科学数据、报告、论文以及相关的学术网页方面非常成功, 并因此获得了 2001 和 2002 年度“最佳专业搜索引擎”。

## 2. 完善现有的通用搜索引擎

目前的通用搜索引擎的 Spider 搜集范围是 Surface Web 页面, 即那些通过超级链接来访问的页面, 对于 Deep Web 所包含的丰富的信息, 几乎不进行访问。其原因是动态内容的特殊访问机制不容易训练 Spider 来自动获取信息。对此, 斯坦福大学的 Raghavan 等 (Raghavan et al. 2001) 给出了一个抽取动态信息的框架的实验原型 HiWE (Hidden Web Exposer)。HiWE 充分利用了动态页面的产生机制, 利用人工辅助的方法解决动态信息的提取。该模型是面向特定任务的, 即针对特定的应用、特定的域和特定用户 (user Profile)。该原型的出现为完善 Web 搜索引擎的功能提供了一个有益的思路。UIUC (Chang et al. 2005) 给出 MetaQuerier 的系统结构和关键构成模块, 其目的是发现 Deep Web 数据库入口和在统一查询接口中访问不同 Deep Web 数据库的原型。

# 第五节 小 结

本章第一节以天网这样一个实际系统由集中式到分布式的过程为例, 说明了基于 Web 信息急速膨胀的需要而产生的对搜索引擎技术发展的要求。提出并设计了可扩展 Web 信息搜集系统结构, 使之达到能够搜集数量不断增长的网页的要求。

在详细介绍可扩展 Web 搜集系统的主要设计思想时, 还对比介绍了集中式搜集系统的设计与实现。它的实际应用就是天网系统。

目前此可扩展搜集系统结构已经实际应用于天网 2.0 系统, 自 2001 年 6 月以来, 共进行了多次大规模的 Web 信息搜集工作, 采用多台搜索机器分布式并行工作。其中 2001 年 11 月 6 日 16 时至 2001 年 11 月 22 日 8 时, 我们的实际运行系统用 12

台 PC 机连接在 100Mbit/s 速度线路上,对全国静态网页(不包括通过提交查询词动态生成的网页)进行了一次搜集。搜集过程从 <http://net.pku.edu.cn> 站点开始,采用一种类似于先宽搜索的策略,直到没有进一步可以搜索的网页。共搜集到网页 47707998 个,涉及 46669 个 Web 站点,其中不重复的网页为 22382623 个,平均每个站点有 47963 个不重复的网页。实际运行和我们的模拟评测结果一致,证明我们并行搜集系统设计高效可行,达到了预定目标。同时我们意识到分布式搜集系统的成功也带来了大量的后续工作:如并行分布式消除重复或转载网页,后台的分布式索引和面向客户的分布式检索,系统管理等。

搜索引擎作为网上信息搜集整理的最直接应用,虽然已经有多家实际系统,但是从分布式系统设计来看,Harvest 搜集系统有诸多地方考虑欠周没有最终实现;从早期的文献看,Google 搜集系统在 URL 分配上采用集中式,而不是分布式,从而会形成系统的瓶颈。从研究状况来看,有关天网的可扩展 Web 信息搜集系统结构设计的论文是较早在国际上发表的应用于搜索引擎实际系统中的阐述分布式系统结构的论文之一(Yan et al. 2001a, Najork et al. 2001)。

分布式系统的动态可配置能力对于系统的适应性、可靠性都有重要的意义。在第二节中论述的两阶段映射模型提供了一种在系统成员变化时协调分布式搜集系统的方法。通过分析,可以采用优化的方案实现两阶段映射模型。当系统进入协调过程中,只需在各个节点间移动网页 URL 对应的 MD5 数据和未访问 URL 列表。每个主控应该维护自己已经搜集网页 URL 对应的 MD5 值。为使系统快速地从不稳定状态进入稳定状态,多播技术对于主控之间的信息更新很有帮助。

至此我们重点介绍了可扩展 Web 信息搜集系统的设计原理与实现方法,并通过一些实验和分析优化了系统的设计,尤其是针对系统动态可配置性的设计。

天网搜索引擎不断在更新,目前正在使用中的分布式搜集系统 P\_Arthur 由 king, queen 和 mosquito 三部分协同工作。它一方面保证了搜索引擎更新所需要的网页数据的获取,同时为中文 Web 网页信息博物馆(网址 <http://www.infomall.cn>)每天获取新出现的网页。

搜索引擎与网页存档工作涉及的网页通常是静态为主的网页,垂直搜索通常需要 Deep Web 中包含的根据用户查询动态生成的网页信息。在本章的最后介绍了 Deep Web 的成因和搜集方法。



## 第七章 网页净化与消重

网页净化和消重是大规模搜索引擎系统预处理环节的重要组成部分。所谓网页净化(noise reduction)就是识别和清除网页内的噪声内容(如广告、版权信息等),并提取网页的主题以及和主题相关的内容;消重(replicas or near-replicas detection)是指去除所搜集网页集合中主题内容重复的网页。建索引一般是在消重后的网页集上进行的,这样就可以保证用户在查询时不会出现大量内容重复的网页。

本章第一节论述了一种 HTML 网页净化与元数据提取的方法,通过它我们可以从一个网页源文件中自动提取网页的一些主要元素,包括网页标识、网页类型、内容类别、标题、关键词、摘要、正文、相关链接等信息。第二节给出了 5 种转载网页的消重算法,通过这些方法可以消除绝大多数主题内容重复的转载网页。

### 第一节 网页净化与元数据提取

今天,当我们浏览 Web,从中获取所需信息的同时,还会常常见大量和我们所关心内容无关的导航条、广告信息、版权信息以及调查问卷等,我们称之为“噪声”内容。有时候,我们可能从这些噪声内容中得到一些意外的惊喜;另一些时候,我们可能不喜欢这些东西消耗人类宝贵的注意力资源。同时,我们观察到噪声内容通常伴随着相关的超链。因此,噪声内容会导致相互链接的网页常常并无内容相关性。这样,网页内容的混乱不仅给 Web 上基于网页内容的工作带来困难,也给基于网页超链指向的工作带来困难。另外,随着 Web 上各种研究与应用的深入发展,仅仅是原始网页内容已经不能满足需求,还要求能够提供便于计算机处理的元数据信息,例如,关键词、摘要、网页内容类别等。然而,现在 Web 上大部分网页仍然是普通 HTML 网页,并不包含必要的元数据。鉴于此,本节讨论一个网页表示模型建立和实现的方法,这一方面使我们能够自动从网页中提取相关的元数据,另一方面也去除了和网页主题内容无关的噪声内容,进而在原始 Web 上搭建一个噪声小、描述清晰、更易于处理和利用的网页信息平台。

在主题搜索领域,大量的广告、导航条等噪声内容会导致主题漂移(topic drift)。这说明传统的主题搜索算法中以网页为粒度构造的 Web 图不够准确,必须深入到网页内部将处理单元的粒度缩小,才能提高内容分析的准确性。在(Chakrabarti et al. 2001)中提出了一套解决方法,首先将网页表示为一棵 DOM 树结构并找到与主题一致性较高的子树,然后对这些子树作特别的处理,从而提高主题提炼的效果。

在 Web 信息检索领域,检索结果的相关性和检索的速度是评价一个 Web 检索系统的两个指标。如果不去除原始网页中的噪声内容,检索系统必然对噪声内容也建立索引,从而导致仅仅因为查询词在某张网页的噪声内容中出现,而把该网页作为结果返回,而网页的主题内容可能和这个查询词完全无关。可以看出,噪声内容不仅使索引结构的规模变大,而且还导致了检索准确性的下降。针对这个问题,(Lin et al. 2002)中提出了一个去除网页中噪声内容的方法,该方法首先依据<table>标签构造网页的标签树,从而依据<table>标签将一张网页规划为相互嵌套的内容块;而后,对于使用同一个模板作出的网页集,找出在该网页集中多次出现的内容,作为冗余内容,而在该网页集中出现较少的内容块就是有效信息块。实验证明该方法是有效的,但该方法必须局限在基于同一个模板的网页集,而 Web 上的网页模板不计其数,该方法显然不够通用。实际上,任意一张网页,人是比较容易区别其中的噪声内容和主题内容的。这说明我们有可能追求自动识别一张网页中的主题内容和噪声内容而不需要依赖于一个网页集合,这样就可以使去除网页噪声内容的方法更加通用和独立。

在网页分类领域,由于噪声内容与主题无关,训练集中的噪声内容会导致各个类别的特征不够明显,而待分类网页中的噪声内容则会导致该网页类别不明确,因而影响了网页自动分类的效果。(Yang 1995, Li et al. 2002)中提出了通过去掉网页中的噪声内容来提高网页分类质量的方法。

在网页信息提取领域,自动识别模式的方法必须要从整个网页中提取模式,而不是只针对主题内容提取。因此,在净化后的网页上作信息提取不仅可以排除噪声信息对信息提取的干扰,提高信息提取的准确性,而且可以使得网页中的结构简单化,提高信息提取的效率。

上述分析我们看到,噪声内容对基于网页的研究工作的影响是普遍而严重的,虽然各个领域采用的方法各不相同,但处理的目的是为了去除网页中的噪声内容,得到真正的主题内容。

另一方面,随着 Web 上研究与应用的发展,单纯的网页内容已经不能满足需求,网页元数据得到越来越广泛的使用。在 Web 信息检索领域,单纯依赖关键词匹配的检索手段过于单一。内容类别、摘要等元数据信息的合理使用,不仅使用户可以从不同的角度进行查询,而且也使得查询的准确性得到提高。而主题搜索、个性化信息服务以及数字图书馆也都强烈的依赖资源的元数据信息。因此,准确且高效的提取必要的元数据是 Web 上各个研究领域面临的重要问题。

在元数据和主题内容的提取方法上,可以从信息提取领域的研究成果(特别是从 HTML 网页中提取语义信息)中得到很多启发。针对从 HTML 网页中提取语义信息,早期的方法是,针对某一类具体网页,人工提取该类网页的内容组织模式。然后,信息提取系统根据该模式从属于该类的网页中提取相应内容(Hammer et al. 1997, Ashish et al. 1997)。对元数据和主题内容的提取可以采用同样的办法,但

这些方法有一个共同的局限性,那就是需要人工提取内容组织模式,这对于内容组织风格繁多的 Web 来说显然是不适用的。因此,在(Weinble et al. 1999)中提出了 5 条启发式规则,综合利用这 5 条规则系统可以自动地发现网页中各个主题信息块(chunk)的边界。(Yang et al. 2001)提出了一种基于视觉相似性来自动分析网页语义结构的方法,该方法首先比较 HTML 网页内容的视觉相似性,然后使用一个模式发现算法来确定这些视觉相似的内容最有可能的组织模式,最后按照该模式将内容重新组合。

通过对上述研究成果的分析,我们发现不同领域的工作存在两个共性:

1) 工作结果的共性。虽然各个领域所做的工作都是为了解决网页复杂化给本领域带来的问题,但各个领域的工作结果中有着共同的部分。譬如,各个领域都需要去除原始网页中的噪声内容;然后在净化后的网页上进行后续工作;很多领域都需要获取网页的元数据信息。即净化的网页和元数据是它们都需要的结果。

2) 工作过程的共性。在获得不同结果的过程中存在着共同的中间环节。譬如,网页分类、摘要的提取以及关键词的选取都需要对文档进行分词操作。而这些中间环节有时是整个工作中效率上的瓶颈。

这些共性启示我们有可能通过归纳不同应用需求中的通用元素,并作为一个模型一次性提取出来,从而对多种应用提供一个统一的支持。可以想象,这样做既便于提高所需信息的质量,又最大限度地避免重复工作带来的时间开销,从而在信息量和复杂性这两个相互制约的因素之间找到一个合理的折中点。

基于这样的思想,本节在参考了 Dublin Core(Dublincore 2003)和 EDA(Encoded Archival Description)(EAD 2002)后,提出了一个包含元数据和内容数据的网页表示模型(称为 HTML\_DocView)。该模型包含这样几项信息元素:网页标识、网页类型、内容类别、标题、关键词、摘要、正文、相关链接。参考上述文献中提出的启发式规则,并结合我们自己对 HTML 网页的统计和观察,本节提出了一套更丰富的启发式规则。在这套启发式规则的基础上,借助传统信息检索领域的方法,结合 HTML 网页的特点,提出了从网页源文件生成其 HTML\_DocView 模型的方法和算法。该方法与前述相关工作相比更为通用,不需要依赖网页模板以及同类的网页集。

本节的方法已被应用到几种具体的实验中;其中,在网页分类实验中,使用 DocView 模型对网页预处理后,分类效果得到普遍的提高。将 DocView 模型应用于搜索引擎的消重过程中,其效果也是明显的(张志刚 2004)。

### 一、DocView 模型

本节中提出的 DocView 模型包括:网页标识、网页类型、内容类别、标题、关键词、摘要、正文、相关链接等要素。其中正文和相关链接要素属于网页的内容数据,而其他 6 项则属于网页的元数据。下面将对模型中的各个要素作详细描述。

网页标识是对 Web 上网页的唯一性标识,在 DocView 模型中使用网页的 URL 作为网页标识。

网页类型是根据网页内容的表现形式进行划分的,在本节中将网页分为三类:有主题网页(topic)、Hub 网页(hub)、图片网页(pic)。其中,有主题网页是指网页中通过文字描述了一件或多件事物,是有一定主题的,如一张具体的新闻网页就是典型的有主题网页。Hub 网页是指专门用来提供网页导向的网页,因而是超链聚集的网页,如门户网站的首页就是典型的 Hub 网页。图片网页是指网页的内容是通过图片的形式体现的,其中文字很少,仅是对图片的一个说明,如某个机构包含图片的人员介绍网页就是典型的图片网页。

将网页分为上述三个类型是因为三类网页在用途和处理方法上存在较大的差别。其中 Hub 网页与其他两类网页的区别在于网页在 Web 上发挥的作用不同,Hub 网页通常不会具体的讲述一件事物,而是提供关于相关信息的链接集。而图片网页与其他两类网页的区别在于处理的方法不同,由于图片网页的内容是通过图片表达的而不是通过文字,因而,传统信息处理领域的方法对图片网页是不够有效的。三类网页间的区别导致很多应用领域都会对它们作适当的区别。

内容类别是从语义上对网页的内容进行分类,它是计算机获取网页语义信息的一个直接手段,在 Web 上的研究领域中有着广泛的使用。它是通过特定的分类器对网页内容分类得到的,依赖于一定的分类体系,本书第十一章将有详细的介绍。

标题、关键词和摘要是概括描述 Web 文档内容的重要的元数据,对于 Web 信息检索等领域的工作有非常重要的作用。

正文是原始网页中真正描述主题的部分,因此,在某些具体应用中用正文代替原始网页更为合理。

相关链接是指在本网页中指向与正文内容相关的网页的链接,而非广告等噪声链接。将正文和相关超链重新组合就得到了净化后的网页。

图 7-1 用 DocView 模型提取的网页要素,图 7-2 是将提取的网页正文和相关超链重新组合成的净化网页。

## 二、网页的表示

网页的表示是网页内容分析的基础,在网页内容分析过程中通常需要两个层次的表示,抽象表示和量化表示。抽象表示是以网页制作规范(如 HTML 规范)为依据和出发点,构造出能体现网页内容结构和内容重要性等信息的表示模型,其目的是充分利用网页制作规范,挖掘出网页中隐含的信息,为后续量化表示提供更多可利用信息。对于 HTML 网页,最常用的方法是构造网页的标签树。量化表示则是从计算机处理的角度出发,利用信息检索领域的技术和从网页中挖掘的隐含信息,生成计算机可以直接用于计算的表示模型,如向量空间模型等。下面对这两个层次的表示方法做详细描述。



图 7-1 用 DocView 模型提取的网页要素

### 1. 抽象表示

HTML 通过定义一套标签来刻画网页显示时的页面。因此,对于 HTML 网页最常用的抽象表示方法是构造网页的标签树。

依据标签的作用可以将 HTML 的标签分为三类:

**规划网页布局的标签:**在视觉上,网页是由若干提供内容信息的区域(我们称之为内容块)组成的,而内容块是由特定的标签规划出的(称之为容器标签),而且容器标签是允许嵌套的。常用的容器标签有<table>、<tr>、<td>、<p>、<div>等。因此,依据容器标签可以将网页表示成树状结构,虽然该树状结构描述的是网页内容的布局结构,但布局信息中隐含着网页内部各部分内容的相关性信息。

**描述显示特点的标签:**在 HTML 标准中定义了一套标签来规范其包含的内容的显示方式(如字体变大、粗体、斜体),我们称之为重要信息标签。常用的重要信息标签有<b>、<i>、<strong>、<h1>、<h2>等十几种。这类标签中的内容通常是网页作者希望引起读者注意的,因此隐含着一定的内容重要性信息。

**超链相关的标签:**超链是 HTML 网页区别于传统文本的最明显的特点之一,表示着网页间的关系,因此整理出超链标签并作合理的分析可以挖掘出网页间的内容相关性信息。

目前,有很多构造标签树的工具(如 W3C HTML lexical analyzer(W3C 1997))

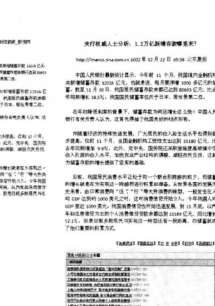


图 7-2 净化后的网页

和 HTML Tidy(HTMLTidy 2004)),它们各有特点,W3C HTML lexical analyzer 有很强的通用性,适合各种标识语言;HTML Tidy 则能够自动发现并修正标签的错误。由于内容分析需要在网页内部计算各个部分之间的相关性以及确定各部分内容的重要性,因此,用传统的顺序整理各种标签的方法构造出的标签树在用于内容分析时并不方便。适合内容分析的标签树强调内容块的概念,倾向于以内容块为单位的内容组织方式。另外,内容分析过程中经常会关心这样一些信息:标签树的规模(节点个数)、每个内容块包含的各种类型信息(如文本、超链或图片)及其数量等。鉴于此,我们自行开发了更适合内容分析的标签树构造工具。

下面简要的描述标签树。给定一篇 HTML 网页,顺序整理出容器标签就得到了对应的标签树的框架。而后,整理每个内容块(对应标签树的一个节点)中的超链标签、图片标签和重要信息标签,并在标签树中对应的节点中记录下来。这样就构造了一棵基本的标签树。对上述基本标签树信息作适当的分析、整理就可以得到内容分析过程中需要的一些描述信息。譬如,依据内容块中词项数与图片数和超链数的比值可以为每个内容块设定一个类型,分为 topic、hub、pic 三种。如果内容块中词项数与图片数的比值小于某个阈值,该内容块就是 pic 类型,如果内容块中作为 anchor text 出现的词项数与该块中总词项数的比值小于某个阈值,该内容块就是 hub 类型,否则为 topic 类型。这样,标签树中每个节点都有类型和属性集两组描述性信息,以及超链集和重要标签集等数据信息。图 7-3 是一个标签树的图例,其中 link\_list 表示该内容块中超链集合;weighty\_tag\_list 表示该内容块中重要标签集合。

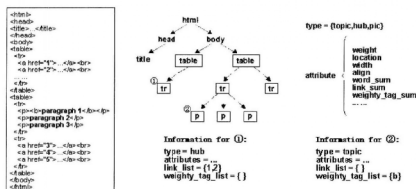


图 7-3 HTML Tree 结构

## 2. 量化表示

### (1) 合理利用网页的特点

在传统的文本处理领域中,一个文本被看做是一个特征项向量( $w_1, w_2, \dots, w_n$ ),其中  $w_i$  是第  $i$  个特征项的权值,  $n$  是特征项的总数。这样,每个文本就被映射到了

向量空间中的一个点,因而向量空间中的点的距离就可以用来衡量其对应的文本的相似性。在量化方法上,对权值的计算,比较常用的是采用  $TF * IDF$  方法。

在量化方法上,我们可以充分的利用 HTML 网页中的重要信息标签信息以及 HTML 网页内容的布局结构。

为了体现重要信息标签中内容的重要性,通常的做法是对重要信息标签中的内容加权值。但重要信息标签中包含的并非都是重要内容,其中的噪声信息非常多,例如,“Tel”、“Fax”、“联系电话”、“传真”、“广告服务”、“前一页”等。我们对此做了这样的统计,从 Web 上随机抓取的 20 000 个网页中,包含在我们定义的重要信息标签中的内容有 9091 条,其中上述的噪声内容(共定义了 22 个)出现了 1200 条,也就是说,重要信息标签中的噪声信息至少占 13.2%。因此,简单地对重要信息标签中的内容加权是不合理的,整理噪声词集合并对重要信息标签中的内容进行过滤,对过滤后的真实重要内容加权值可以避免噪声扩大化。

由于网页中的标签结构是对页面布局的描述,我们不难得到这样的结论:如果某个内容块中存在真实重要信息,那么这个内容块的重要性也相对较高;如果一个内容块的重要性较高,那么这个内容块的外层嵌套块的重要性也相对较高。可以看出,导致网页中内容块重要性增加的是包含真实重要内容的重要信息标签。基于这个结论,我们给网页中每个内容块赋予一个权值,用来表示这个内容块的重要性,并提出内容块权值的传递规则(我们称其为权值传递规则)。由于内容块与标签树中节点是一一对应的关系,以下对权值传递规则的描述统一使用标签树的节点而不使用内容块。

权值传递规则:

- 标签树中每个节点的初始权值为 1。
- 每个重要信息标签都有一个影响因子。如果标签树某个叶子节点中存在重要信息标签并且重要标签中的内容是真实重要内容,那么累加重要信息标签的影响因子,得到的和就是该叶子节点的影响因子。没有出现重要标签的叶子节点的影响因子为 1。
- 对于每一个叶子节点,如果影响因子为  $\lambda$  且  $\lambda > 1$ ,则该叶子节点的权值变为当前值的  $\lambda$  倍,它的父节点以及父节点下的其他子树中的节点均变为当前值的  $\sqrt{\lambda}$  倍,然后以该父节点为变化源,按照上述规则再向外扩展一次。每一次扩展过程中,遇到父节点为  $\langle body \rangle$  或父节点权值超过预定上限就结束整个权值传递过程。过程如图 7-4 所示。

不难证明,“权值传递规则”有以下两个性质:

**性质 7-1** 对于初始的标签树,无论从哪个节点开始、以什么顺序执行“权值传递规则”,标签树最终的权值结果都是相同的。

**性质 7-2** 如果初始标签树中叶子节点影响因子的分布不同,那么标签树最终的权值结果一定是不同的。

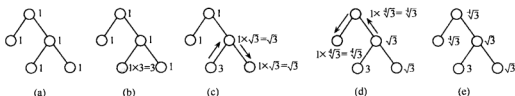


图 7-4 内容块权值传递过程

其中,性质 7-1 是保证规则正确的基本条件,性质 7-2 则说明,“权值传递规则”可以保证:初始标签树中叶子节点影响因子的分布与最终标签树中权值结果是一一对应的。另外,可以证明,“权值传递规则”的两个性质与权值向上传递的层数是无关系的。

## (2) 适合内容分析的 HTML 网页量化表示

内容分析过程中的处理对象是网页中的内容块,对于内容块的表示,特征向量方法同样是适用的。但在具体的量化方法上有所不同。一个最重要的区别在于,内容分析过程中侧重的是一张网页内部各个内容块之间的相似度比较,而不是网页间的相似度比较。因此,在特征项权值的计算方法上,我们更侧重特征项在一张网页内部的重要性,而不是特征项在一个文档集合上基于统计的重要性。基于上述分析,我们使用公式(7-1)来计算特征项权值。

$$w_i = \frac{\sum_{j=1}^{BN} BWeight_j \times BT_{fj}}{\sqrt{\sum_{i=1}^n (\sum_{j=1}^{BN} BWeight_j \times BT_{fj})^2}} \quad (7-1)$$

其中, $BWeight_j$  表示内容块  $j$  的权值,它的值由一个内容块中的重要标签来决定的; $BN$  表示网页中内容块的总数; $n$  表示网页中不同关键词的总数; $BT_{fj}$  表示关键词  $i$  出现在内容块  $j$  中的词频。

## (3) 量化表示方法分析与改进

分析上述的特征项权值公式可以知道,如果网页中没有出现重要标签信息,则所有内容块的权值均为 1,那么公式(7-1)就变为完全依赖于词频的计算方法。统计发现,只有 19% 的网页中有重要标签,这就是说,公式(7-1)对大多数网页而言,是简单使用词频来衡量特征项的重要性;但是,文档中很多高频词并不是真正重要的。基于上述分析可以知道,必须要对高频词作特殊的处理。

所谓“高频无关词”,是指虽然在文档中词频很高,但却没有主题描述能力和区别能力,例如,“中国”、“可以”。在基于词频的权值计算方法中,该类词的权值将会很大;另外,“高频无关词”很容易出现在重要标签中,因而对重要标签中的信息加权也使得这种词的权值很大。因此,在内容分析之前去掉“高频无关词”,既可以提高网页内容表示的准确性,又能减少网页向量中的维数,提高效率。



“高频无关词”最明显的特征是：在大量的文档中都以高频词的角色出现。基于这个特征，我们可以通过词频和文档频率确定某个文档集合的“高频无关词”集。如果使用网页集合并且该集合的规模足够大，那么就可以得到近似 Web 上的“高频无关词”集。实验数据表明，“高频无关词”与非“高频无关词”在作为高频词出现的文档频率上有很大的差别。因此，我们可以依据该跳变的位置确定高频无关词集。

### 三、提取 DocView 模型要素的方法

对 Web 上的网页，我们根据其网页类型可以将它们分为三类：有主题网页、Hub 网页和图片网页。针对三类网页的信息提取算法各不相同，因此在对网页进行深入分析之前首先要判断网页的类型。为此，我们首先描述这三类网页的特征及判断方法，然后将对面向有主题网页的模型提取算法进行详细的讨论，最后简要的介绍面向 Hub 网页和图片网页的算法。

#### 1. 网页类型判断方法

在视觉上，大多数网页是容易区分类型的，因为三种类型的网页有着较为明显的视觉特征。在有主题网页中通过成段的文字描述了一件或多件事物，虽然也会有图片和超链，但这些图片和超链并不是网页的主体。图片网页中内容是通过图片体现的，而文字仅仅是对图片的一个说明，因而文字不多。Hub 网页通常不会描述一事物，而是提供指向相关网页的超链，因此，Hub 网页中超链密集。

虽然视觉上判断网页的类型是比较容易的，但让计算机自动做到这点却不容易。下述的量化方法可以在绝大多数的情况下准确的识别网页的类型。网页都是有一定布局的，比如分左右两边或是中间和边缘。网页作者通常将重要的内容放在网页的中间部分，而边缘部分内容的重要性相对较低，这也是符合人的浏览习惯的。因此，依据网页中间区域的内容判断网页的类型是相对合理的，而网页中内容的位置信息在本节中构造的标签树中是通过内容块的属性记录下来的。本节前面提到，在构造标签树时，依据内容块中词项数与图片数的比值以及内容块中词项数与 anchor text 中词项数的比值将网页中的内容块分为 topic、hub 和 pic 三个类型，基于内容块的类型，我们可以使用网页中间区域 hub 内容块包含的词项数与网页中间区域词项数的比值来判断网页是否为 hub 类型。同理，使用网页中间区域 pic 内容块包含的词项数与网页中间区域词项数的比值可以判断网页是否为 pic 类型。实际效果表明，该方法判断网页的类型是较为准确的。

#### 2. 有主题网页的信息提取算法

该算法以一组启发式规则为指导，首先提取出网页的正文信息，然后以正文信息为基础，提取 DocView 模型中其他的要素。过程如图 7-5 所示。下面按照各个要素

的生成过程分别描述。

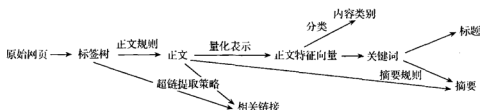


图 7-5 有主题网页 DocView 模型生成过程

正文：

一篇有主题网页中的正文通常是用成段的文字来描述，中间通常不会加入大量的超链，而非正文信息通常是伴随着超链出现的。基于此，我们提出了正文选取的规则（称为正文规则）。

正文规则：有主题网页中，如果一个内容块是 topic 类型的，则该内容块中的内容为正文的一部分。

依据正文规则，深度优先遍历标签树并依次记录 topic 类型的内容块，就得到该网页的正文，也就是该网页的主题内容。

关键词：

关键词选取的依据是特征项的权值，因而特征项权值的合理计算是正确提取关键词的保证。以标签树为基础，结合 HTML 网页的特点以及提出的量化方法，可以按照下述过程得到网页主题内容的特征向量。

使用图 7-6 所示算法得到特征项向量后，我们可以用两种策略决定选取关键词的数量。

```

1: for 标签树中的每个正文块 CBi do
2:   if 该块中存在重要信息标签信息 then
3:     检查重要信息标签中的内容是否在噪声词集中出现
4:     if 不在噪声词集合 then      //为真实重要信息
5:       将重要信息标签的影响因子累加到该内容块的影响因子上
6:     end if
7:     if 该内容块的影响因子大于 1 then
8:       提出的权值传递策略在标签树中传递权值
9:     end if
10:  end if
11: end for
12: 计算各个特征项的权值
  
```

图 7-6 计算网页特征项权值的算法

1) 绝对数量策略。首先定义好 DocView 模型中关键词的个数  $\alpha$ , 严格选取权值最大的  $\alpha$  个特征项作为该网页的关键词。

2) 相对数量策略。该策略中不需要规定要选取的关键词的个数, 而是依据特征项权值的绝对大小。该策略首先定义了一个阈值  $\beta$ , 而后计算所有特征项权值的算术平均值  $\text{avg}$ , 选取特征项中权值大于  $\text{avg} * \beta$  的作为该网页的关键词。这种策略虽然会导致每个网页中被选取关键词的数量不均, 但它却可以更准确地提取关键词。

#### 内容类别:

内容类别是通过正文分类得到的。网页的量化表示是网页分类过程中必不可少的阶段, 而在前面关键词提取过程中已经得到了正文的特征向量, 于是直接使用正文特征向量进行分类可以节省网页量化过程的时间开销, 这正是将共性需求的信息一次性提取的优势之一。仅对网页的正文分类有效的排除了噪声内容的干扰, 从而提高了分类的准确性。我们使用的是北京大学网络实验室开发的分类器(详见第十一章)。

#### 标题:

HTML 网页中, 网页的标题由 `<title>` 标签标识。通过统计我们知道, 94.12% 的网页是有标题的, 但在这些标题中, 有很多是如下标题: “Untitled Document”、“New page”、“welcome”、“欢迎访问”。这其中有的是网页制作工具为新创建的网页赋予的初始名称, 有的是网页制作者较为常用的网页标题。它们是没有任何网页描述能力的, 因而并不是合格的标题。针对没有标题或者使用上述无描述能力标题的网页, 我们从关键词集合中选取权值最高的作为网页的新标题。

#### 摘要:

摘要的提取基于这样的事实: 文章都是按内容分段组织的; 阅读者通常是根据一段文章中某几个子句来得到该段文章的大意, 而这几个子句的选择通常是通过扫描某些关键词来定位的。因此, 如果可以自动识别文章中不同的段落, 那么基于上述得到的关键词, 就可以得到能够模拟读者浏览文章过程的摘要提取算法。

#### 识别文章段落:

HTML 网页中的结构信息是对网页版面的描述, 这使得自动识别文章的不同段落成为可能。在正文提取部分已经得到了网页的正文, 在网页的标签树中, 所有正文块也构成了一个树状结构, 称之为正文树。在正文树中, 首先找到所有叶子节点的最近共同祖先节点作为正文根节点。正文根节点的各个子节点对应的正文块就是正文的不同段落。段落识别过程如下图所示。图 7-7 中的 `<tr>` 内容块就是正文根节点, 其下面三个 `<p>` 内容块就是三个段落。

#### 基于段落的语句提取:

以正文的段落为单位, 在各个段落中定位网页的关键词并累加关键词的权值作为关键词所在语句的权值; 最后在每个段落中限量选取权值大的语句, 就组成了网页的摘要。该方法得到的摘要不能保证摘要中的语句之间有上下文关系, 但能做到简

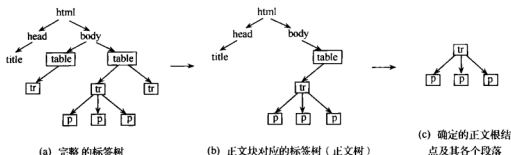


图 7-7 正文段落识别过程

短的摘要能覆盖整个文章的内容。

相关超链:

在超链选取的过程中,我们基于这样一个假设:网页中的超链在网页排版时通常按照主题聚集,换言之,相同主题的超链在网页中的位置是相近的,通常放在一个最里层的内容块(该内容块中不再包含其他内容块)中或并列的几个最里层内容块中。这就意味着我们可以以内容块为单位对超链进行取舍。对于超链的选取,我们实验了两种策略。

#### (1) 基于 anchor text 的超链选取策略

anchor text 是对超链所指向网页简短、概要的说明,在一定程度上体现了被指向的网页的内容。基于 anchor text 的超链选取方法是通过比较每个 hub 类型内容块中 anchor text 集合与正文的相似度来决定该块中链接的取舍(图 7-8)。

```

1: 计算网页正文对应的特征项向量  $\varphi$ 
2: for 网页中的每个叶子内容块  $CB_i$  do
3:   if  $CB_i$  是 hub 块 then
4:     计算  $CB_i$  对应的特征项向量  $\varphi_i$ 
5:     计算  $\varphi$  与  $\varphi_i$  的相似度 similarityi
6:     if similarityi >  $\beta$  then //  $\beta$  为相似度阈值
7:       保留  $CB_i$  中的 URL
8:     else 不保留  $CB_i$  中的 URL
9:   end if
10: end if
11: end for
  
```

图 7-8 基于 anchor text 的超链选取算法

通过这一算法,我们可以对 hub 类型内容块中的超链进行取舍,而其他类型内容块中的超链通常是对正文中某些信息的详细说明,因而其他类型内容块中的超链通常是内容相关的。因此,整理上述算法保留的 Hub 类型内容块中的超链和其他类型内容块中的超链就构成了整篇网页的相关超链集。

## (2) 基于分类的超链选取策略

基于分类的超链选取方法是通过判断一个 Hub 类型内容块中某个超链(通常是第一个)指向的网页与本网页正文的类别是否相同来决定该块中所有链接的内容相关性。该方法可以有效地解决上述方法中 anchor text 信息过少的不足,而且实验结果证明,该方法确实比基于 anchor text 的方法准确,但需要动态的从 Web 上抓取并分类,因而时间开销很大。

## 3. Hub 网页和图片网页的提取算法

与有主题网页相比,Hub 网页和图片网页有着自己的特点。Hub 网页中,网页的主题就是提供超链,超链本身就是正文。因此,相关超链信息对 Hub 网页来说是没有意义的。图片网页中,网页的主题内容就是图片和描述图片的信息,而描述图片的信息很多情况下是以超链的形式出现的。因此,图片及其周围的信息(包括超链)都是网页的正文。在处理过程中基于如下的假设:在网页的布局上,重要的信息通常是放在中间,网页两边信息的重要性相对较弱。因此,对于 Hub 网页和图片网页,我们可以将网页中间区域的内容块作为网页的正文,而周围的内容块则通过与正文比较相似性来决定取舍。

## 四、模型应用及实验研究

### 1. DocView 模型在网页自动分类中的应用及实验分析

网页与传统文本的一个重要区别是网页内容的随意性,这就导致网页内容中的噪声内容很多,因此,在网页分类过程的开始首先对网页作适当的净化,可以在一定程度上改进分类的准确性。将 DocView 模型中正文要素和相关超链要素重新组合就得到了净化的网页。在本实验中,我们以一个现有的分类器作为基准,提取基准分类器的训练集和测试集中网页的 DocView 模型,并用模型中正文要素和相关超链要素组合成的新网页替换原始网页,从而形成净化的训练集和测试集。然后,通过对净化后的训练集学习得到新的分类器,并用净化后的测试集进行测试。通过比较新分类器与基准分类器的测试结果,证明模型中正文要素和相关链接要素提取的正确性以及用它们代替原网页的合理性。

#### (1) 实验数据集

实验中用北京大学网络实验室开发的分类器作为基准分类器,该分类器支持的分类体系中共有 733 个类别,分为三层,其中顶层类有 12 个,该分类器的训练集和测试集共有 15 570 个网页,这些网页就是按照上述分类体系组织的。

如果 DocView 中正文要素和相关超链要素提取的正确性足够高,那么把 DocView 中正文要素和相关超链要素组合起来生成的新网页将有效地消除噪声内容,因此用新网页代替原网页后的新训练集会使得各个类别的主题更为明显,而新测试集

中网页的类别也更为清晰。所以,基于新训练集和测试集的测试结果也会得到改进。

## (2) 实验评测标准

对分类效果的评价采用传统的查准率、查全率、 $F_1$ 值。

$$\text{precision}_i = \frac{\text{分类到类别 } i \text{ 并且分类正确的文档个数}}{\text{实际分类到类别 } i \text{ 的文档总数}}$$

$$\text{recall}_i = \frac{\text{分类到类别 } i \text{ 并且分类正确的文档个数}}{\text{实际属于类别 } i \text{ 的文档总数}}$$

$$F_{1i} = \frac{2 \times \text{precision}_i \times \text{recall}_i}{\text{precision}_i + \text{recall}_i}$$

## (3) 实验结果及其分析

图 7-9 是新旧分类器的性能比较,横轴表示不同的类别编号,对应上面式子中的下标  $i$ (最后一项是关于所有 12 类的平均值),表 7-1 是图 7-9 中类别编号对应的类别含义。

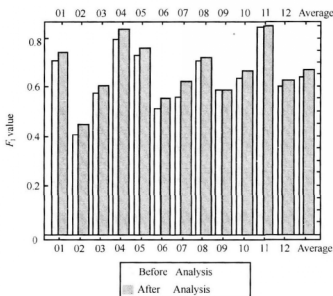


图 7-9 网页净化前后分类效果对比

表 7-1 类别编号对照表

类别号	01	02	03	04	05	06	07	08	09	10	11	12
类别名称	人文与艺术	新闻与媒体	商业与经济	娱乐与休闲	计算机与因特网	教育	各国风情	自然科学	政府与政治	社会科学	医疗与健康	社会与文化

通过图 7-9 我们看到,所有类别的分类结果均比原来有所改善。由于我们并没有对分类器的特征项提取算法和分类算法作任何改进,所以,图 7-4 所示的提高完全是从网页本身的改进得到的。因此可以得到两条结论:

- 净化过程中,没有出现明显的信息遗漏。
- 广告等噪声信息确实得到有效的去除。

由于人工选取训练集和测试集的网页时已经做到尽量选取正文信息多、噪声信息少的网页,因此,网页净化在实际应用中的效果要比该实验的结果更为明显。

## 2. DocView 模型在网页消重中的应用及实验分析

在本实验中,我们将 DocView 模型应用于预处理环节中的网页消重<sup>①</sup>。消重是指将搜集到的网页中的镜像或转载网页去掉的过程(镜像网页可以理解作为一种特殊的转载网页),在消重后的网页集上建索引再提供服务可以保证用户查询时不会出现大量内容重复的网页。由于大量的转载网页并不是对原始网页的简单拷贝,而是将要转载的内容放在新的模板中再提供服务。因此模板中的内容就会干扰消重程序对转载网页的判断,从而导致错误消重。常见的错误消重有以下两种情况:

- 相同的内容,由于放在了不同的模板中导致应该被消掉但实际上被消重程序判断为非转载网页而保留。
- 不同的内容,由于放在了相同的模板中导致不应该被消掉但实际上被消重程序判断为转载网页而消掉。

从实际系统中也可以看出,模板因素是导致消重不够准确的一个主要原因。鉴于此,本实验中首先提取网页的 DocView 模型,然后将模型中的正文要素取代原始网页作为判断转载网页的依据。

### (1) 实验数据集

实验中使用三组网页:基准网页集、转载网页集、非转载网页集。

基准网页集是用来模拟 Web 上被转载的原始网页,该集合中的每一个网页都在一个或多个满足特定条件的转载网页在转载网页集中,也存在一个或多个满足特定条件的非转载网页在非转载网页集中。

转载网页集是用来模拟 Web 上的转载网页,该集合中存放基准网页集中每个网页对应的转载网页,这些转载网页满足如下条件:与对应的基准网页使用不同的模板。该集合用来测试上述情况一。

非转载网页集是用来模拟 Web 上使用相同模板但内容不同的网页,该集合中存放基准网页集中每个网页对应的非转载网页,这些非转载网页满足如下条件:与对应的基准网页使用相同模板而且内容属于同类。要求与基准网页内容属于同类是因为属于同类的网页容易被误消,所以使用同类的网页能更好地说明问题。该集合用来测试上述情况二。

理想情况下,转载网页集中的网页应该全部被消掉,而非转载网页集中的网页应该全部保留。因此,我们可以通过计算一个消重算法对转载网页集和非转载网页集

<sup>①</sup> 关于网页消重算法见本章第二节。

消掉的网页个数来评价该消重算法。

我们人工为基准网页集、转载网页集、非转载网页集三个集合选取了网页,网页内容覆盖体育、娱乐、新闻、科技、社会、财经、教育、医学、IT、游戏等领域,目的是为了实验网页的正文内容覆盖面足够广从而避免模板内容与网页正文内容有特殊关系。三个集合中共有 79 个网页,其中基准网页集有 26 个,转载网页集有 26 个,非转载网页集有 27 个。

### (2) 实验思想

由于消重的目的是消去主题内容相同的网页,而 DocView 模型中的正文要素正是去掉网页中模板内容后的主题内容,因此,在使用相同的消重算法的前提下,用 DocView 模型中的正文要素代替网页原文参与消重,消重准确性应该有所提高。

### (3) 实验结果及分析

表 7-2 中的数据是使用同一种基于关键词的消重算法但基于不同内容的消重结果。表中的数据是被消掉的网页数。

表 7-2 消重实验结果

消重结果	转载网页集(共 26 个)	非转载网页集(共 27 个)
基于网页原文的消重结果	1	0
基于净化网页的消重结果	20	0

从上表中的数据可以看到,在基于网页原文的消重方法中,由于模板因素的干扰,导致消重算法将模板不同的转载网页作为不同网页而没有消掉。而用 DocView 模型中的正文要素代替网页原文参与消重,可以很好地克服模板因素的干扰,从而提高消重准确性。

## 第二节 网页消重算法

如前所述,我们粗略地将内容完全相同的网页称作镜像网页,主题内容相同的网页称作转载网页。就消除主题内容重复的网页而言,我们完全可以把镜像网页看做转载网页的特例来处理。由此,所谓网页消重就是指去除网页集中转载网页的过程。

国际上对转载文档消重算法的研究最初主要是针对大型文件系统的,后来又被拓展应用于数字化图书馆项目和搜索引擎系统。美国 Arizona 大学的研究人员采用计算文档的重叠程度的方法来发现一个大型文件系统中的相似文件(Manber 1994)。Stanford 大学的研究人员开发了 SCAM (Stanford Copy Analysis Mechanism)原型系统用于发现相似的数字化文档,后来在对其消重算法作了改进之后应用于 Google 搜索引擎系统。几乎所有的上述消重技术都基于这样一个基本思想:为每个文档计算出一组指纹(fingerprint),若两个文档拥有一定数量的相同指纹,则认为这两个文档的内容重叠性较高,也即二者是内容转载的。



由于上述系统的应用目标不同,它们计算文档指纹的方法以及在如何度量两个文档的相似程度方面有较大差别。文献(Shivakumar et al. 1998)采用了一种对全文分段签名的算法。这种算法把一篇网页按一定的原则分成  $N$  段(如每  $n$  行作为一段),然后对每一段进行签名(即计算指纹),于是每一篇文档就可以用  $N$  个签名后的指纹来表示。对于两篇文档,当它们的  $N$  个签名中有  $M$  个相同时( $m$  是系统定义的阈值),则认为它们是互为转载的网页。该算法使用对〈文档标识(DocID),段标识(ChunkID),指纹(Fingerprint)〉三元组排序的方法避免了对所有网页作两两比较,使算法复杂度有所降低。但是该算法的空间复杂度和时间复杂度仍然是相当大的,若应用于海量的搜索引擎系统(通常包含上亿个 Web 页面),仍然难以取得理想的效果。我们结合基于关键词匹配的搜索引擎系统的特点,提出了 5 种网页消重方法,实验表明这些方法的效果很好,目前已被成功地应用于北大天网搜索引擎系统,用于消除转载网页(谢正茂 2003)。

## 一、消重算法

### 1. 算法基础

当前比较成功的搜索引擎系统大多是基于关键词匹配和结合向量空间模型来完成用户的检索请求的。典型的系统包括 Google 和天网系统。通常这类系统在对已抓取回来的网页进行分析时,要提取网页中出现的关键词和摘要信息,并以关键词作为网页的特征项。

天网系统在搜集并分析一篇网页时,提取并记录了网页中出现的关键词,同时根据公式赋予每个关键词一个权值,这些关键词的权值构成一个向量空间,可以用来表示该网页。另外,我们还从网页中提取了 512 字节的有效文字(指用户实际访问该网页时能看到的文字,在 html 和其他格式的网页中,有一些用户看不到的文字,它们告诉浏览器该执行什么样的操作及如何显示网页,包括字体、颜色、排版等信息)作摘要。当用户查询时,摘要同网页的标题、URL 等信息一起显示给用户,供用户了解网页的内容,选择感兴趣的进行浏览。

### 2. 算法描述

考虑到基于关键词匹配的搜索引擎系统的特点,结合使用网页的向量空间模型,我们提出了 5 种网页消重算法,用于快速、有效地发现 Web 上的转载网页。下面逐一介绍这几种算法。在以下的描述中,用  $P_i$  表示第  $i$  个网页,其权值最高的前  $N$  个关键词构成的特征项集合为  $T_i = \{t_1, t_2, \dots, t_m\}$ ,其对应的特征向量为  $W_i = \langle W_{i1}, W_{i2}, \dots, W_{im} \rangle$ ,其摘要用  $\text{Abstract}(P_i)$  表示,前  $N$  个关键词拼接成的字符串用  $\text{Concatenate}(T_i)$  表示,而先对前  $N$  个关键词按字母序排序后再拼接成的字符串用  $\text{Concatenate}(\text{sort}(T_i))$  表示。另外,用  $\text{MD5}(X)$  来表示字符串  $X$  的 MD5 散列值,用

Mirror( $P_i, P_j$ )表示  $P_i$  和  $P_j$  互为转载网页,用  $A \Rightarrow B$  表示“若  $A$  成立则  $B$  成立”。

算法 1

$$(\text{MD5}(\text{Abstract}(P_i)) = \text{MD5}(\text{Abstract}(P_j))) \Rightarrow \text{Mirror}(P_i, P_j)$$

算法 2

$$(\text{MD5}(\text{Concatenate}(T_i)) = \text{MD5}(\text{Concatenate}(T_j))) \Rightarrow \text{Mirror}(P_i, P_j)$$

算法 3

$$\left. \begin{aligned} &(\text{MD5}(\text{Concatenate}(T_i)) = \text{MD5}(\text{Concatenate}(T_j))) \\ &\left( \frac{|W_i - W_j|^2}{|W_i|^2 + |W_j|^2} \right) < \delta, \quad \text{其中 } 0 \leq \delta \leq 1 \end{aligned} \right\} \Rightarrow \text{Mirror}(P_i, P_j)$$

算法 4

$$\left. \begin{aligned} &(\text{MD5}(\text{Concatenate}(\text{sort}(T_i))) = \text{MD5}(\text{Concatenate}(\text{sort}(T_j)))) \\ &\left( \frac{|W_i - W_j|^2}{|W_i|^2 + |W_j|^2} \right) < \delta, \quad \text{其中 } 0 \leq \delta \leq 1 \end{aligned} \right\} \Rightarrow \text{Mirror}(P_i, P_j)$$

算法 5

$$(\text{MD5}(\text{Concatenate}(\text{sort}(T_i))) = \text{MD5}(\text{Concatenate}(\text{sort}(T_j)))) \Rightarrow \text{Mirror}(P_i, P_j)$$

### 3. 算法分析

可以看出,我们设计的第 1 种算法采用了对网页摘要求 MD5 散列值的方法,当两个网页的散列值(占 16 个字节)相同时,就认为二者是互为转载的。由于 MD5 算法的严格性保证了当两个网页的摘要内容有一个字节不同时,其散列值就不同,这样就使得本来应作为转载处理的却没有被确定为转载网页。为此,我们又基于向量空间模型理论,提出了第 2 种和第 5 种算法。第 5 种算法表明,当两个网页的权值最高的前  $N$  个关键词集合相同时就认为二者是互为转载的网页。第 2 种算法比第 5 种要严格一些,它不仅要求两个转载网页的前  $N$  个关键词相同,其顺序也是一致的(按权值排序),因而第 2 种算法有可能会漏掉一些转载网页。

算法 2 和算法 5 都只是要求转载网页的前  $N$  个特征项相同,没有考虑到这些特征项所构成向量的夹角大小。算法 3 和算法 4 则分别在算法 2 和算法 5 的基础上分别考虑了两个网页特征向量的相似度。但向量相似度的计算并没有使用夹角余弦值来定义,因为它只度量了两个向量的夹角大小,而没有考虑向量的长度。我们认为只有当两个向量的夹角小,同时其长度相差也小时,二者才是相似的。针对这一点,又设计了判断两个向量相似度的方法,即算法 3 和算法 4 的第二个条件

$$\text{SIM} = \left( \frac{|W_i - W_j|^2}{|W_i|^2 + |W_j|^2} \right)$$

可以看出, SIM 能够同时兼顾向量的夹角和长度两个因素。当两个网页内容毫不相关时(即它们的关键词集合没有交集),  $W_i$  与  $W_j$  垂直, SIM 的值为 1。当两个网页相同时, SIM 为 0。当两个网页相似而不相同时, SIM 的值介于 0 和 1 之间,于是 SIM

的值成为判断两个网页相似度的标准。另外,类似于算法 5 是对算法 2 的条件放松,算法 4 也是对算法 3 的放松。

后四种算法都对向量空间模型理论作了较大的简化。首先,我们只从网页中出现的所有关键词组成的  $M$  个特征向量提取了前  $N$  个( $N < M$ ),这把理论模型的限制放松了。之所以可以这样做是因为:

1) 特征向量的前  $N$  个分量绝对值大,基本能确定特征向量的方向。取较少的关键词能减少算法的复杂度。尽管有可能降低其准确度,降低多少,后面的实验将对其作出评测。

2) 转载网页的制作人,对网页稍加改动变成相似网页时,不能改变其基本意思。而网页的基本意思一般通过其中出现的高频词来反映。后面的  $(M - N)$  个词出现的次数为 1 或 2,相对而言,这些词的出现是不稳定的,当使用这些词来判断相似网页时,反而会漏掉一大批相似网页。

其次,后 4 种算法都要求前  $N$  个关键词组成的集合要相同。这却把理论模型的限制加强了。这主要是由于对算法复杂度的考虑,判断两集合交集大小需要先求出它们的交集。求交集运算的复杂度较大,而把一百万网页两两求交集,其  $10^{12}$  量级的运算量是我们不敢提及的。我们只能考虑用 MD5 算法对集合签名,实际上就是对关键词序列签名,来表示集合的相同与不同。签名算法有极高的敏感性,作用对象稍有不同就会给结果带来很大的差异,并且不可能从签名差异的大小来判断原签名对象差异的大小。作这样的简化后,有可能出现这样的情况,位置在  $N$  附近的词在排序上出现的微小变动,如第  $N$  个词与第  $N + 1$  个词位置互换了,本来是两篇相似度很高的文章,可能会被我们的算法漏掉。这对算法的影响到底有多大,我们仍需通过实验来评测。

## 二、算法评测

### 1. 评价方法

我们为网页消重算法设计的评价指标包括算法复杂度、查全率和准确率三个方面,其中算法复杂度又包括时间复杂度和空间复杂度。在本节中,查全率是指消重算法所发现的转载网页占总网页的百分比,而准确率反映了算法所发现的转载网页中有多少是真正的转载网页。假设要处理的网页数为  $N$ ,后 4 种算法使用的关键词个数为  $M$ ,每个关键词的权值占 4 个字节,MD5 摘要占 16 个字节,则算法 1、2 和 5 的空间复杂度约为  $(16 \times N)$ ,算法 3 和 4 的空间复杂度约为  $(N \times M \times 4 + N \times 16)$ 。可以看出这 5 种算法的空间复杂度都很小。本节将重点考察算法的时间复杂度、查全率和准确率。时间复杂度和查全率将直接基于天网系统 2000 年 5 月份的数据并运行算法来得到。由于算法输出的转载网页太多,我们无法一一判断每个网页是否是真正的内容转载网页,因而对于准确率的计算我们采用了如下的估算方法:先将算法

的输出结果(即算法求出的转载网页)分为  $n$  段,在每段中对镜像随机取 100 个采样,用人工确认的办法得到每 100 个采样的准确率,最后用这  $n$  个准确率的平均值来作为算法的准确率。

## 2. 实验结果

在 2000 年 5 月上旬,我们使用天网的搜集子系统从国内的 2 万多个 Web 站点上搜集了 1 182 899 个网页,对于每个网页我们都保存了其摘要、关键词及其权值等信息,依此作为我们的实验对象。这 5 种算法运行的机器是一台 PC 机,配有双 CPU,内存为 256MB,硬盘 36GB,运行的操作系统为 Turbo Linux 6.0。

基于上述实验环境,我们分别使用上述 5 种算法来消除转载网页。当关键词个数  $N$  取 10、向量偏差度阈值  $\delta$  取 0.01 时,得到了第一组实验结果如表 7-3 所示。其中“总和”是指只要某个网页被五种算法中的任一种检测为转载,它就被确定为转载。实际上这里的“总和”等价于算法 1 和算法 5 的“总和”,因为算法 2、3 和 4 发现的转载网页都能够由算法 5 所发现。算法 1 与其他算法的差别较大,这是由于它们所选取的判断对象的不同造成的,前者选用了 512B 的摘要,而后者用的是一组关键词。从表 7-3 可以看出,这五种算法的准确率都很高,而其中第 5 种具有最高的查全率,而准确率的损失又不是太大,我们认为效果是最好的。同时,也可以看出“总和”的查全率要比 5 种算法中的任意一种好得多,而其准确率也很高。这启示我们可以结合使用算法 1 和 5 来消除转载网页。

表 7-3 当  $N=10, \delta=0.01$  时 5 种算法的查全率和准确率

算法编号	总网页数	转载数	查全率/%	准确率/%
算法 1	1 182 899	199 763	16.89	98.11
算法 2	1 182 899	212 914	18.00	98
算法 3	1 182 899	210 475	17.79	98.03
算法 4	1 182 899	219 841	18.58	97
算法 5	1 182 899	225 919	19.10	97
总和	1 182 899	265 752	22.47	97.09

由于算法 3 和 4 的效果受向量偏差度阈值  $\delta$  取值的影响,我们令  $\delta=0.1, N=10$  重复了上述实验,得到的结果如表 7-4 所示。可以看出  $\delta$  的取值对算法的查全率和准确率影响不大。

表 7-4 考察  $\delta$  的取值对算法 3 和 4 的影响

算法编号	总网页数	转载数	查全率/%	准确率/%
算法 3	1 182 899	211 718	17.90	98
算法 4	1 182 899	222 151	18.78	97

关键词个数  $N$  的取值是影响后 4 种算法效果的一个关键因素,因而我们针对效果最好的第 5 种算法,通过取不同的  $N$  值,得到了查全率的变化曲线,如图 7-10 所示。从图中可以看出,当关键词取得极少时,镜像算法的查全率很高,这无疑是用极低的准确率作代价的。关键词慢慢增多,查全率迅速下降,当选取到 9 个或 10 个关键词时,这种下降趋势变得很平缓,同时准确率达到一个比较高的水平。当选取到 20 个以上的关键词时,有些低频词被选取,一部分转载网页的差别被反映出来,于是查全率有较大下降。随着选取的关键词继续增多,有差别的转载网页全部被认为是不同网页,剩下的完全相同网页使查全率不再随选取关键词的多少而变化。我们得到的结论是,关键词取 10 个左右最恰当,在较高的准确率的基础上,获得了最大的查全率与最小的运算量(主要是指关键词排序和签名运算)。

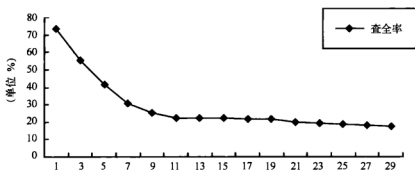


图 7-10 查全率随选取关键词个数的变化

### 3. 与现有算法的比较

为了与现有的算法进行比较,我们把文献(Shivakumar et al. 1998)的实验结果(主要包括查全率和运行时间)列于表 7-5 中。我们的实验结果列于表 7-6 中。其中文献(Shivakumar et al. 1998)的实验平台为带双 CPU 的 SUN UltraSPARC 工作站,内存为 256MB,对换工作区大小为 1.4GB,机器运行的操作系统为 SunOS 5.5.1。可以看出其运行环境是优于我们的。基于关键词的消重算法有一个优点,就是只用一次签名就可以很好的判断某网页是否为转载,这样对  $N$  个网页计算其网页转载的复杂度为  $O(N)$ 。通过比较表 7-5 和表 7-6 的数据,可以看出,算法 1 和 5 平

表 7-5 分段签名算法的时间复杂度及性能

分段方法	网页总数	所用时间	查全率/%
全文作一段	24 000 000	2760 分钟	17
每四行作一段	24 000 000	2940 分钟	22
每两行作一段	24 000 000	3240 分钟	22

表 7-6 基于关键词的各算法的时间复杂度及性能 ( $N=10$ ,  $\delta=0.01$ )

算法编号	网页总数	所用时间	查全率/%
算法 1	1 182 899	5 分钟	16.98
算法 5	1 182 899	4 分钟	19.10
总和	1 182 899	9 分钟	22.47

均处理一个网页的时间总和不到“分段签名算法”的十分之一。我们的算法在时间上有明显的优势。另外,从这两个表可以看出,算法 1 和 5 总和的查全率要好于分段签名算法的,这表明消重的效果也略好于文献(Liu et al. 2000)的分段签名算法。

当然我们的算法也有其局限性。如算法 1 要求事先知道各个网页的摘要信息,而后 4 种算法则要事先知道每个网页的关键词。但是目前绝大多数的搜索引擎是基于关键词匹配的,系统本身已为我们找出文章的摘要和关键词。而对于数字化图书馆项目而言,大多数数字化文档的摘要和关键词也是事先知道的,因而我们的算法仍然是适用的。

### 第三节 小 结

本章第一节提出了一种 HTML 网页净化和元数据元素的提取方法,通过这种方法我们可以从一个源文件中自动提取网页的一些主要元素,包括网页标识、网页类型、内容类别、标题、关键词、摘要、正文、相关链接等信息。第二节描述了 5 种网页消重算法,并使用天网系统对之进行了评测,体现了这些方法的优越性。

Web 存在大量的镜像和转载网页,如下为比较常见的几类:

1) 政府部门的法令、通告。例如,公安部颁布的《计算机信息网络国际联网安全保护管理办法》在国内的网站上有 120 个镜像。

2) 重大新闻、热点新闻。例如,《中欧 WTO 谈判取得进展但未达成协议》有 75 个镜像。

3) 技术文档。如 TUCOWS WinSock utilities 有 100 个镜像,Micorsoft Internet Information Server 2.0 Manuls 有 90 个镜像。

4) 提供一定服务的网站和网页。例如,“黄金书屋”的镜像有 30 个,而列出了国内各所大学 URL 的网页有上千个。

除了采用本节的消重算法来提高搜索引擎系统的输出质量外,还可以研究以网页的镜像度为参考来判断网页的重要度,进而提高 Web 搜集系统的搜集质量和效率。

## 第八章 高性能检索子系统

以 Google 为代表的商业搜索引擎获得了很大成功。到 2004 年 4 月, Google 已经索引了全世界 42 亿个页面, 每天接受上亿次查询请求。但是商业搜索引擎的核心技术属于商业机密, 在激烈的竞争环境下不会公开。而在研究领域, 因为受到条件的限制, 对大规模通用搜索引擎系统的技术探讨也较少。在第二章我们介绍过, 搜索引擎包括搜集子系统, 预处理和服务子系统三大部分。有时候为方便起见, 将建立索引和提供服务放在一起, 称为检索子系统。搜集系统研究如何更快速抓取更多高质量网页的相关技术, 检索系统研究如何进行网页文档索引, 为用户提供高性能的检索服务。后者主要建立在信息检索领域的相关技术之上, 同时根据 Web 自身的特点也发展了一些新技术。文献(Brin et al. 1998)是 Google 在斯坦福大学的原型系统的一个较全面技术介绍, 其中重点介绍了其检索系统的设计与实现, 验证了使用链接分析技术可以有效提高搜索引擎检索效果。和传统的信息检索系统相比, 大规模搜索引擎的检索系统面临许多新的挑战。

本章以天网 Web 服务检索系统为基础, 分析搜索引擎的检索系统设计与实现的基本技术; 在搜索引擎中采用的两种数据结构: 倒排索引结构和平面位置索引; 为了减少存储倒排索引的空间, 并提升查询处理的速度而采用的技术, 包括倒排索引压缩、索引剪枝、混合索引技术和倒排文件缓存机制。

### 第一节 检索系统基本技术

#### 一、系统设计与结构

搜索引擎检索系统的设计围绕检索效率和检索效果这两个指标展开。对一个成功的搜索引擎来说, 首先必须具有相当高的检索效率。由于通用搜索引擎是面向大众的, 其信息需求的重要性参差不齐, 绝大多数可以说是“随心所欲”的, 其价值不值得等待很长的时间, 因此一个响应迟缓的系统只能意味着较少的用户。按一般的习惯, 搜索引擎对用户查询的响应时间应该不超过秒级, 这相对于搜索引擎需要处理的海量网页数据而言是一个挑战。而如何提高搜索引擎检索效果, 更是人们不断研究的课题, 但它是要在保证检索效率的前提下才有意义。因此, 信息检索领域有一种观点, 认为搜索引擎的检索技术相对于最新的信息检索研究成果是一种倒退。如果仅从检索效果上看, 确实如此, 但由于搜索引擎面临的效率压力, 使得在实现上往往需要在效率和效果之间折中, 而不一定采用效果最好的技术。同时, 有统计表明在

Web 搜索环境下,用户普遍使用短查询、不做查询优化(Silverstein et al. 1998, Spink et al. 2001, Wang et al. 2001),这些特点也是搜索引擎提高检索效果面临的主要困难(因为用户向系统提供的信息太少)。但在另一方面,传统信息检索只从文本内容上计算文本和查询的相关程度,而 Web 环境下,除了网页文本数据,还有大量其他信息可以为这一相关性的计算提供辅助支持,比如网页内的 HTML 标记,URL,链接关系,Anchor text,网站目录数据等。如何有效利用这些信息是搜索引擎提高检索效果的一个重要途径。

天网的检索系统设计原则有两个:一是追求系统效率和可扩展性。二是力图通过一个集成的框架结构,能够有效地把各种有利于改善检索效果的技术集成起来,如图 8-1 所示。这样一个框架结构体现在三个方面:①文档表示。对一个网页文档可以有多种角度的表示方式,包括索引词、半结构化的元数据以及全局的网页属性。②用户信息需求的类型识别,以求能为不同类型的信息需求选择最佳的检索方式。③不同检索排序方式得到的结果的融合。

在图 8-1 中,方框表示检索系统,在服务点(SE ServicePoint)接受用户的查询请求(User InfoNeed)。用户请求经过检索代理(Retrieval Agent)分类,进行检索策略的选择,调用索引服务提供的相应检索机制来完成检索。通常,搜索引擎提供的最基本检索方式是基于关键词的布尔查询(Boolean OP)。但通常用户输入的查询为自然语言词语或者短语,并不是一个布尔表达式。一般情况下,搜索引擎默认用户的输入查询词之间为与(AND)关系。为了提高检索效果,有些搜索引擎也采取查询词扩展(query expansion)和相近(proximity)计算技术,并用这些计算的结果来驱动后台的结果提取过程。Google 成功的使用了链接分析技术(Kleinberg 1998, Page et al. 1998),为每个网页赋予一个全局的权值(PageRank 值)来表示网页的重要程度。网页的这种全局属性的检查在图 8-1 中由 GlobalProperties 模块执行,除了 PageRank,还可以包括根据权威的网站目录数据、用户反馈或人工编辑等方式得到的网站权值。Meta 是元数据查询的执行模块,可以包括时间、文档格式、站点名称、分类类别等各种网页元数据,针对网页数据的信息提取技术可以融合到这一模块中。天网在中文网页自动分类方面有一个研究小组,其网页文本自动分类技术已经应用在天网目录服务和检索中(冯是聪 2003)。在检索系统框架中,网页文档的分类类别起着重要的作用。利用 Meta 模块返回的网页类别信息,Retrieval Agent 可以进行类别聚合,把相同类别的网页集中显示给用户,这样一种方式可以更好的组织检索结果,改善检索效果。Semantic Constrains 模块是语义的约束检查模块,它建立在对网页文本中特定语义关系识别的自然语言处理技术之上,是实现回答自然语言问题的必要技术,第十二章将要介绍的“天网知名度”(Fame 2004)就是这样一种技术成功应用。

天网检索系统的具体实现同样基于信息检索技术。

首先是排序算法和检索模型的选择。在图 8-1 的框架结构中,检索系统的相关



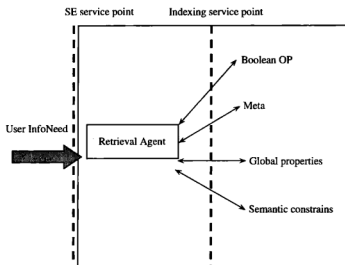


图 8-1 检索系统集成框架结构

性排序由多种因素综合决定。这其中，最基础的排序建立在信息检索的布尔模型和向量空间模型基础上。在 BooleanOP 模块中，首先执行布尔查询，得到的结果作为候选文档集合，然后按向量空间模型的相似度算法计算各个文档与查询的相似度，结果作为排序的基础。最后由 RetrievalAgent 综合其他模块返回的信息，再进一步排序。典型的一种情况是，当查询词在 AnchorText 或者 Title 中出现时，把全局属性里的 PageRank 值与文档的相似度权值通过线性组合方式相加得到最后的排序权值。排序采用一种分级算法，分为三个级别：查询词的邻近关系运算结果；查询词出现的位置，包括 Title、AnchorText；相似度权值与其他的权值，如全局属性的 PageRank 权值。各种权值通过线性方式组合起来(Lei et al. 2001)。

其次是索引的实现技术。天网检索系统采用倒排文件索引。对于大规模文档数据，倒排文件是经过大量实践检验的一种高效率的索引组织方式，能够很好的支持多种检索模型，提供高性能的检索。人们对倒排文件的组织和检索效率做了大量的研究工作，文献(Moffat et al. 1996, Witten et al. 1994, Navarro et al. 2000)对天网的索引实现有重要的影响。(Moffat et al. 1996, Witten et al. 1994)重点在倒排索引的压缩，(Navarro et al. 2000)在倒排索引的随机访问技术，它们都被应用到天网的索引系统中。由于搜索引擎的责任是索引不断变化着的海量网络信息，倒排文件的组织还需要在检索效率和更新效率上进行折中。一般倒排索引的索引项数据用链表方式分块存放有利于提高更新效率，但这会降低检索效率；反之，索引项数据连续存放有利于检索，而不利于更新。天网检索系统以检索效率为主要优化的目标，索引更新采用部分索引重建的方式。

整个检索系统采用分布式系统结构。搜索引擎的海量网页数据索引无法在单台

机器上集中完成,分布式系统是解决数据规模和系统可扩展性问题的基本方法。天网检索系统的系统结构如图 8-2 所示。

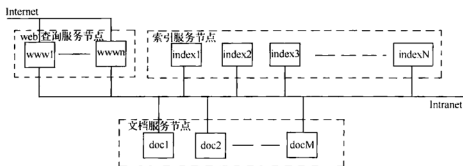


图 8-2 天网 WWW 检索分布式系统构架

现在运行的检索服务系统共使用 20 台 PC(PIII733/1GB), 其中一台为 WWW 查询服务器, 其余 19 台为索引服务器, 文档服务节点和 WWW 查询服务器使用同一机器。文献(Lu 1999)通过性能仿真实验对分布式信息检索系统的可扩展性进行了深入研究。我们注意到, 在学术界当人们谈及“分布式信息检索”时指的是研究不相交数据集在分布环境下的数据集选择和检索问题, 和这里谈的搜索引擎检索的分布式系统结构不是一回事, 但其提出的部分复制技术有很好的参考价值。文献(Tomasic et al. 1993)研究了倒排索引的物理组织对分布式查询的性能影响, 认为最好的数据分布方式是“host index organization”, 也就是每一个文档的全部索引项应该都分布在同一台处理机上。这种数据组织方式不仅可以最大限度降低节点间通信开销, 而且由于索引节点之间相互独立工作, 整个查询系统有很好的容错性。天网的分布式检索系统设计就以此作为出发点(赵江华 2002)。

前面我们从总体上描述了天网检索系统的设计思想和系统结构, 下面两部分分析天网的索引创建和检索实现技术。

## 二、索引创建

对一个中文搜索引擎, 索引创建不仅仅是一个高效的倒排算法, 它还包含许多重要的方面: 索引词的选择, 中文分词、编码识别与转换、网页净化、强健的页面分析等。

### 1. 索引词选择

索引词的选择是检索系统实现的一个重要环节。现代搜索引擎普遍使用全文索引技术, 即网页文档中所有词都参与索引。理想的索引词应该是表达文档内容的语义单位, 即语言学里的词语, 是那些专指义, 而实际意义无法由组合成分相加得到的最小语言单位。但实际系统中中文文本必须通过自动分词程序的处理, 分割成为独

立的分词单位,再从分词结果中选择索引词。自动分词算法有两大类,普遍采用的方式是基于词典的分词方法,这一方法效率高,但分词精度受词典规模制约;另一种是基于统计语言模型的方法,可以发现一些新词。实际应用是两种方法的不同程度的组合。

除了中文分词外,对英文单词、数字、英文缩写词、特殊专有词的识别由一个词法分析器完成,通过不断完善词法规则,就可以支持如“C++、C#、AT&T”这样的一些特殊词。英文单词统一转换为小写,但不作词根和词形变换。

## 2. 网页预处理

创建索引需要对网页进行分析,其中编码转换是一个重要步骤。Web 上的网页包含多种字符集和编码,搜索引擎的索引系统必须对它们转换,采用统一字符集和编码方案。UNICODE 是一种兼容性较好的字符集选择,而且可以使用不同的编码方案,如 UTF8、UTF16 等。但由于程序移植和编程习惯上的困难,天网目前仍然使用了 GBK 为系统的内部编码,这对于主要面向中文的搜索引擎已经足够了,但如果进一步考虑国际化,UNICODE 应该是更好的选择。常见的中文编码包括简体中文的 GBK、GB2312、ISO-2022-CN、GB18030,还有繁体中文的 BIG5、BIG5HKSCS 和 EU-CTW。在 GNU 的 GLIBC 库中有对这些编码的转换支持,但是对于简体、繁体转换是简单的基于字一字映射,准确度不高。在网页分析中,识别网页的编码方式也比较繁琐。按 HTML 的规范(W3C 1999),页面内容的编码依次由 Web 服务器返回的 HTTP 头信息中的 charset 字段;网页中 meta 标签里的 charset 属性以及每个网页元素的 charset 属性确定。但在实际情况中,网页存在许多编码设置错误的现象,尤其是那些由简繁体自动转换而生成的中文网页。这时需要有一个自动识别编码的模块,按统计的方法自动识别网页的正确编码。

此外,大量网页中存在不符合 HTML 规范的错误,这要求网页分析模块十分健壮。同时许多网页中存在大量无用的信息,如广告、导航条等,这一现象在大型网站使用相同模板的网页中普遍存在。这些网页噪声对用户的信息检索没有意义,因此不应该被包含到索引范围内。

## 3. 索引创建算法

天网检索系统采用带位置信息的词级全文索引。系统采取了按站点划分网页数据的分布式方案,各个索引节点相互独立,索引创建过程在每个节点上独立进行。采用两趟的内存倒排创建算法,依次为每个小文档集倒排,最后执行多路归并,生成总的倒排文件。主要步骤如下:

1) 页面分析。按 HTML 语法规则分析网页标签结构、调用中文分词和英文词法分析器提取索引词。分析过程中记录每个索引词的文档频率 df 和在文档内的词频 tf,通过散列表转换为索引词编码,保存得到词典文件(lexicon file),并保存页面

分析的结果到临时文件。

2) 按统计得到的索引词的  $tf$  和  $df$  属性,可以估计出对应倒排项数据的长度,以此预申请整个文档集合倒排需要的内存空间。重新读取页面分析保存结果的临时文件,在内存中执行倒排,把结果保存到临时倒排文件中。

3) 对生成的多个临时倒排文件,执行多路归并,压缩编码,输出得到最终的倒排文件。

在索引创建过程中,页面分析,特别是中文分词为主要时间开销。算法的后两步相对很快。这样创建算法的优化集中在中文分词效率上,而没有采用效率更高的倒排创建算法(Heinz et al. 2003)。

### 三、检索过程

天网分布式检索系统执行查询时,由 WWW 查询服务器通过多播把用户输入的查询串发送给每一个索引节点。各索引节点独立在本机上执行查询,再把检索结果中排序最前的  $K(K=100)$  个结果返回给 WWW 查询服务器,在  $K$  值控制合理情况下,可以把返回结果数据包控制在一个以太网数据帧大小内,使系统具有很小的网络通信开销和延迟。WWW 查询服务器上的 RetrievalAgent 负责结果数据的收集、合并、重新排序,并访问文档服务器、提取摘要,格式化生成查询结果页面返回给查询用户。

文献(Wang et al. 2001)通过分析搜索引擎用户查询日志,提出并在天网 WWW 查询服务器上实现了检索结果缓存。使用 LRU 缓存替换算法,缓存容量为 500 个检索结果时,缓存命中率能达到 60% 左右,有效提高了系统整体性能。

在天网的分布式检索结构中,系统性能瓶颈最终在索引节点(赵江华 2002)。通过实验发现,索引节点的检索效率瓶颈在于磁盘系统的性能,检索算法中对倒排文件中查询词对应的倒排项数据读取是检索效率优化的重点。首先采用的是系统级优化措施。注意到操作系统中 I/O 系统的实现特点,采用 C 函数库提供的带缓冲的文件访问接口效率最差,操作系统提供的底层文件访问接口  $read/write$  效率较好,而使用内存文件映射或者直接设备访问可避免多次的内存拷贝问题,从而大大提高 I/O 访问的效率(Newman 2001)。其次再通过对倒排文件的组织优化,通过减少每次访问倒排项数据的数量和访问次数来提高检索效率。天网系统中采用了三种基本技术:索引压缩、随机访问的索引组织和重要索引词单独索引。

#### 1. 索引压缩

倒排索引压缩可以减小倒排项数据长度。在检索过程中可减少内存和 I/O 带宽的使用,但同时要对压缩数据解码,增加了 CPU 时间耗用。实际系统中,I/O 是系统的瓶颈,而且 CPU 和 I/O 之间性能差距还在不断扩大,所以索引压缩技术作为一种有效提高检索效率的技术被普遍采用。

倒排索引压缩的方法基于“游程编码”，增量整数序列被变换为差分序列。组织倒排索引文件，可以把倒排项中的文档号和出现位置编号，都按递增序排列，这样可以通过“游程编码”变换，把大整数序列变换成较小的整数序列，再选取一种整数编码方案实现高效的倒排项数据压缩。

文献(Witten et al. 1994)中给出了多种变长编码方法，其中 Golomb 编码是压缩效率较优的一种。文献(Williams et al. 1999)比较了多种编码方法的编码和解码效率。在搜索引擎应用中，检索效率是主要优化目标，而索引数据的空间占用相对并不重要。变长编码有解码慢的缺点，天网系统实际采用字节对齐的定长编码 ByteCode。实验测试得到 ByteCode 和 Golomb 的平均压缩比率分别为 0.3359 和 0.2635，解码时间两者的比例为 1 : 6。

## 2. 随机访问的索引组织

文献(Navarro et al. 2000)提出了对倒排索引的索引项建立二级索引，使得可以随机访问倒排项数据块。在一般情况下，这一技术可以减少倒排项数据的访问量，但同时可能增加 I/O 访问的次数。在采用这一技术时需要确定随机访问倒排项数据块的大小，在节省 I/O 带宽与 I/O 访问的次数的开销之间取得最好的折中。具体来说，小数据块访问会带来更多次系统调用，带来更多次的寻道时间消耗；大数据块访问读入冗余的数据，带来过多的数据传输时间消耗。根据磁盘访问性能分析的实验表明，使用较大的数据块系统性能较好，因此天网检索系统目前采用 32KB 为最小块单位。

在二级索引之外，倒排项数据还使用数据块自索引技术(Moffat et al. 1996)。选择 32KB 为二级索引块大小，每 32KB 的位置信息记录一个开始文档号。每块数据内部使用 512B 作为自索引的段长，使用 ByteCode 压缩编码。自索引技术不减少 I/O 数据访问量和访问次数，但使得检索算法在处理倒排项数据时，可以跳过一些压缩的数据块，节省处理时间。

## 3. 重要索引词单独索引

对重要索引词单独索引，这样可以产生一个小的倒排索引文件，控制其大小能保存在内存中，如果有相当的查询在这个小索引文件中获得足够的返回结果，则查询结束；当检索得到的结果不足时，才去访问磁盘上的整个倒排文件。通过这一方式，系统可以节省大量磁盘的访问开销，大大提高效率。

这一技术有效应用的前提是小索引中查询得到的结果文档在整个倒排文件的查询结果集合中排序在最前面，否则会降低系统检索质量。这一点可由排序算法保证。被选择的重要索引词包括 Anchor text, Title 还有利用天网文档模型技术(Zhang et al. 2004)提取的正文摘要中的词。

## 第二节 适于查询的网页索引结构

在大型综合搜索引擎中,适于查询的网页索引结构通常采用倒排索引结构。最近流行的一种不同思路的组织索引结构的方法是平面位置倒排索引(flat position index),其优点是可以通过提高短语查询的速度。位置索引文件把整个文档集看成一个词序列,每个词都有一个其相对于序列开始的绝对位置值。而每个被索引词对应的倒排表只包含这个词在序列中出现的位置。同时,用数组保存每个文档在词序列中的边界。

### 一、倒排索引结构

倒排索引通常由词典(dictionary)和倒排表(inverted index)两部分组成,如图 8-3 所示。词典保存索引文档集中所有出现的词及这些词对应倒排链(posting list)在倒排表中数据的偏移信息。通过查找词典,可以得到一个词是否被索引,以及其对应数据在倒排表中的位置。倒排表由索引词的倒排链组成。一个索引词的倒排链保存了出现这个词的文档号列表,以及一些关于这个词在文档集中的统计信息,例如出现的次数,出现的位置等。按照倒排链中的倒排项(posting)的组织方式划分,主要有以下两类倒排索引:①按文档号递增排序的索引(document sorted index)(Anh et al. 2006, Zobel et al. 2006)。②按词频排序(frequency sorted index)

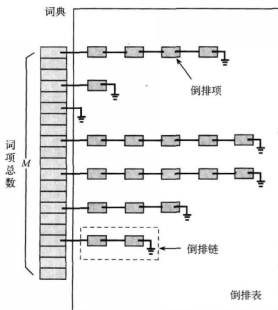


图 8-3 倒排索引结构示意图

(Persin et al. 1996)或 impact 值排序的索引(impact sorted index)(Anh et al. 2001, Anh et al. 2006, Strohmaier et al. 2007)。impact 值是索引词在一篇文档中重要程度,往往用一个值较小的正整数表示(Anh et al. 2001)。词频繁排序或者 Impact 值排序的索引结构相对于文档号递增排序的索引主要有以下局限性:①对于 impact 索引,其每个索引词对应的文档的分数在索引阶段已被固定,不能在检索时调整。②其把倒排链分成的多个不同权值块,不利用提升倒排索引的压缩率。③不能很好支持短语查询或者临近查询(proximity search)(Anh et al. 2006)。文献(Ding et al. 2011)表明,按文档号排序的索引经过一定的优化后,其查询速度可以比 Impact 值排序的索引快。所以现在的搜索引擎通常使用按文档号递增排序的索引结构来处理用户提交的查询。

在按文档号递增排序的索引结构中,倒排链中的每个倒排项通常是一个三元组:

$$\langle d, f_{t,d}, [\text{pos}_1, \dots, \text{pos}_{f_{t,d}}] \rangle$$

其中,  $d$  是文档号,  $f_{t,d}$  是词在文档内的词频,  $\text{pos}_i$  是词在文档中的出现位置。文献(Anh et al. 2006, Zhang et al. 2008, Yan et al. 2009, Ding et al. 2011)讨论了各种文档号列表,词频列表与位置列表的组织方式及其对查询的影响。一种高效的倒排链组织方式是按块进行组织,一个块可以是变长的也可以是定长的,例如每个块由 128 个倒排项组成。这种组织方式的好处:①每个块都可以单独的压缩与解压缩。②同类型的数据能被一块儿压缩以达到更大的压缩率,同时对词频与位置信息的数据可以在需要时被解压。③对于每个块,可以保存一些块信息,可以快速的估计块中文档的相关性(Chakrabarti et al. 2011, Ding et al. 2011)。图 8-4 是按块组织方式的倒排链的一个示意图,每个块包含 128 个倒排项。在图中,一个索引词的倒排链由多个块组成。在倒排链的开始,存储块的跳查表(skip table)信息。跳查表由跳查指针(skip pointer)组成。一个跳查指针通常是一个二元组  $\langle d, p \rangle$ , 表示在倒排链的  $p$  位置有一个以文档  $d$  开始的块。每个块的开始存着一些元信息,例如压缩算法名,块包含的文档数等。在解压数据的时候可以先把文档号进行解压,然后根据查询的需要来肯定是否解压频率和位置信息。

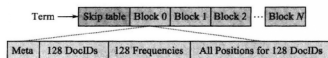


图 8-4 按块组织的倒排链的结构

## 二、平面位置索引

本节主要讨论平面位置索引的详细结构及其在这种结构上的查询处理。

### 1. 索引结构

在平面位置索引中,它把整个文档集看成一个词序列,每个序列中词都有一位偏移地址。例如,在图 8-5 中,整个文档集包含 5 个文档,每个文档相当于被拼接成一个长序列。索引主要由三部分组成:字典,倒排文件及文档边界数组。字典的结构跟传统的倒排索引中的词典结构类似,包括索引词的基本信息(如文档频率)和倒排表在倒排文件的偏移。在倒排文件中,每个词的倒排表与传统的索引有较大的差别。在这种索引结构中,只包含词出现位置信息的列表。例如词 A 对应的倒排表是<1, 6, 14, 17, 22, 28>。而在传统索引中,词 A 的倒排表是[(1, 2, <1, 6>)(2, 2, <4, 7>)(3, 1, <4>)(4, 1, <5>)]。因此各个词的倒排表中,需要存储的数据比传统的索引要少。文档边界数组用于保存在文档集中的各个文档的起始地址。在传统的索引中,如果检索模型需要文档长度,如 BM25 模型,其通常会保存文档长度数组。在位置索引中,通过文档边界数组号就可以得到文档长度,所以不需要再保存文档长度数组。

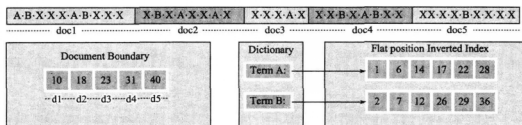


图 8-5 位置索引的结构

在这种索引结构中,如果文档集很大,那么这些文档拼接后产生的序列的长度也会变的很长。例如,在 GOV2 数据集中,其包含 2500 万文档,总的序列长度近 230 亿,超出来 32 位数能表示范围。一个直接的想法是用 64 位数到表示每个词的偏移,也就是倒排表中的每个倒排项是 64 位数,但是这会直接导致在解压数据时,需要多传输 1 倍的数据量从内存来 CPU。另外,由于倒排表中的数字是进行  $d$ -gap 的,真正需要用 64 位的情况很少。所以在我们的实现中,当索引的文档集很大,把整个文档集看成一个大数据序列,然后对这个大数据序列进行切分,使它变成多个子序列,每个子序列都有一个编号,这个编号很小,用一个 byte 表示。每个词的偏移只在子序列中计算,这样用 32 位数就可以表示其在子序列中偏移。所以每个词在总序列中的位置由子序列编号和其在子序列中的偏移组成。这种方法会使得在整个序列空间中有些位



置不包含词,但这不影响最后的计算。这种切分的主要思想就把文档集分成多个子文档集,每个子文档集由多个文档组成,但其拼接的长度不会超过 32 位数的表示范围。

## 2. 短语查询处理

短语查询要求其所找到的文档中包含这个短语。在传统的索引中,短语查询就是对每个查询词的倒排表求交的过程。主要有 term-at-a-time 和 document-at-a-time 这 2 种处理方法(Anh et al. 2004)。实验表明,在查询词比较少的情況下用 document-at-a-time 的处理方式比较快,而在查询词多于 4 个的时候,term-at-a-time 的处理方式会比较快。使用传统词级别的索引结构处理短语查询的算法已在(Anh et al. 2004; Garcia-Molina 2008)详细的介绍,在这里我们主要介绍使用位置索引处理短语查询。其也可以分成 term-at-a-time 和 document-at-a-time 的处理方式,算法 8-1 和算法 8-2 分别描述了这两种处理方式。在 term-at-a-time 的处理过程中,一般按照每个查询词的 postinglist 长度(CF, collection frequency)从短到长进行处理,开始把最短的 postinglist 当作候选短语位置集合 C,然后用其他的 postinglist 来检索这个候选集中的每个元素是否满足成为短语的要求。最后由候选位置集生成短语出现的文档集 D。

算法 8-1 在位置倒排索引上按 term-at-a-time 的方式处理短语查询。

1. open the postinglist for query term  $t_1$  with the smallest CF;
2.  $C \leftarrow \text{copy-list}(t_1)$ ;
3. **for**  $i \leftarrow 2$  **to**  $|q|$  **do**
4.  $d \leftarrow \text{distance between } t_i \text{ and } t_1 \text{ in query}$ , open the postinglist for term  $t_i$ ;
5. **for** each candidate  $c$  **in**  $C$  **do**
6. set  $p \leftarrow \text{skip-to-value}(t_i, c+d)$ ;
7. **if**  $p > c+d$  **then**
8. remove  $c$  from  $C$ ;
9. **if**  $|C| = 0$  **then**
10. **return** no answers;
11. set document identifier  $d \leftarrow 0$ , begin and end document boundary  $b_1 \leftarrow 0$ ,  $b_2 \leftarrow 0$ , frequency  $f \leftarrow 0$ , position list  $P \leftarrow \{\}$ ;
12. set  $n_1$  and  $n_2$  the number of query terms before or after  $t_1$  in query;
13. **for** each  $c$  **in**  $C$  **do**
14. **if**  $c + n_2 > b_2$  **then**
15. **if**  $P$  is not empty **then**
16. set  $s \leftarrow \text{score}(d, f, P)$ ; set  $f \leftarrow 0$ ; set  $P \leftarrow \{\}$ ; set  $D \leftarrow D + \{(d, s)\}$ ;
17. set  $(d, b_1, b_2) \leftarrow \text{find-docid-bound}(c)$ ;
18. **if**  $c - n_1 < b_1$  **then**

19. **continue;**
20.  $\text{set } f \leftarrow f+1; \text{ set } P \leftarrow P + \{c \cdot n_i\};$

算法 8-2 在位置倒排索引上按 document-at-a-time 的方式处理短语查询。

1.  $\text{set document identifier } d \leftarrow 0, \text{ begin and end document boundary } b_1 \leftarrow 0, b_2 \leftarrow 0, \text{ frequency } f \leftarrow 0, \text{ position list } P \leftarrow \{\};$
2. **for**  $i = 1$  **to**  $|q|$  **then**
3.   open the posting list for term  $t_i$
4.    $\text{set } p_i \leftarrow \text{next-value}(t_i)$
5. **while** all list have pointers remaining **do**
6.    $\text{set } p \leftarrow \max\{p_i \mid 1 \leq i \leq |q|\}, \text{ set } k \leftarrow i \text{ where } p = p_i$
7.   **for**  $i \leftarrow 1$  **to**  $|q|$  **then**
8.      $\text{set } p_i \leftarrow \text{skip-to-value}(t_i, p \cdot k + i)$
9.   **if**  $p_i > p$  **then**
10.      $\text{set } p \leftarrow p_i, k \leftarrow i$
11.   **if**  $\min\{p_i \mid 1 \leq i \leq |q|\} = p \cdot k$  **then**
12.     **if**  $p \cdot k + |q| > b_2$  **then**
13.       **if**  $P$  is not empty **then**
14.          $\text{set } s \leftarrow \text{score}(d, f, P); \text{ set } f \leftarrow 0; \text{ set } P \leftarrow \{\}; \text{ set } D \leftarrow D + \{(d, s)\};$
15.          $\text{set } (d, b_1, b_2) \leftarrow \text{find-docid-bound}(p);$
16.     **if**  $p \cdot k < b_1$  **then**
17.       **continue;**
18.   **for**  $i \leftarrow 1$  **to**  $|q|$  **then**
19.      $\text{set } p_i = \text{next-value}(t_i)$
20. **return**  $D$

在这两个算法中,最关键的是  $\text{skip-to-value}(t, p)$  和  $\text{find-docid-bound}(p)$ 。 $\text{skip-to-value}(t, p)$  语意是查找  $t$  的倒排表,直到其位置大于或者等于  $p$ ,如果  $p$  被找到了,那么这个位置被保留在候选集中,如果找到的值比  $p$  大,那么这个位置应该从候选集中去除。 $\text{Find-docid-bound}(p)$  的语意是在边界数组中,查找位置  $p$  对应的文档号和这个文档的开始与结束的边界。

在使用平面位置索引处理短语查询时,可以把查询分解成以下 5 部分:①从外存中加载每个查询词的倒排表时间;②解压倒排表的时间;③查找短语出现位置的时间;④把短语出现的位置转成文档号的时间;⑤对文档打分的时间。

在此索引中,每个词的倒排表中只包含位置信息,相对于传统的词级别的索引少了文档号和文档内频率这两种信息。因此,需要存储的数据相对于传统索引少很多,所以在存储空间上比传统索引开销更少。读数据倒排表数据的时间在查询处理中往往占大部分。

对于解压数据的时间,如果平面位置索引和传统索引都不用跳查表,那么这两种

结构需要解压全部的倒排表数据,而前者的倒排表包含的数组比传统索引少,所以解压数据所花的时间比传统索引少。另外,虽然在平面位置索引中有值会比较大,但对于一些最新的压缩算法如 PForDelta 对不同大小的数字的解压速度基本一样。

在传统的索引中,短语查找需要两个步骤:第一步通过文档号找到可能出现短语的文档,第二步通过读取查询词在文档中的位置信息,查找短语出现的位置。这种方法找到包含短语的文档数量比第一步得到的候选文档数量少。在平面位置索引中,短语查询也分成两步:第一步找短语出现的位置,第二步把这些位置转成文档号。在算法一和算法二中,可以看出需要转文档号的次数与查找的文档数量相等。对位置与文档号的转换,其速度也是可以非常快的完成,见下面“缓存敏感的查找表”部分。

对于文档的打分的时间,由于使用的特征一样,所以这两种索引所花的时间基本一样。

### 3. 缓存敏感的查找表

在短语查询处理中,使用平面位置索引进行短语查询时,需要把出现短语的位置通过文档边界数组转成对应文档号。当文档边界数组非常巨大(千万量级),如何高效地把位置转文档是一个非常关键的问题。这个问题可以转化为一个更常见搜索问题:给定一个正整数  $p$  和一个有序数组  $T$ ,找出在  $T$  中第一个大于等于  $p$  的元素的下标  $k$ 。

常见的查询算法,如二分查找可以很好的解决这个问题,但是如果搜索的数组很大,其搜索效率并不高效。因为现代计算机体系结构中,CPU 与内存的性能差距越来越大。缓存被引入以减小 CPU 和内存两者之间性能差距。CPU 通常按照 cache line 的大小获取数据,其大小通常为 32 字节或 64 字节。当 CPU 需要访问的数据已在缓存时,所需要的代价很小,我们通常称这种情况为缓存命中。当数据不在缓存时,所需要的代价很大,这种情况我们通常称缓存缺失。这时需要从内存中获取数据,因为其访问的基本单位是 cache line,也就是说即使访问一个字节的的数据,至少会加载 32 字节的数据。如何有效的利用这些多读的数据成为提升系统性能的关键点之一。在二分查找的算法中,并没有很好的使用 cache 这个特性,所以在查找时,缓存缺失率很高,平均为  $\log_2 n$ 。现在很多缓存敏感的查找算法如 m-array 查找树, CSS-tree (cache sensitive search tree) 和 CPSS-tree (cache/pagesensitive search tree) (Bookstein et al. 1994) 充分利用计算机结构中这一特性,查找速度也往往比二分查找快,但是这些算法的复杂度仍为  $O(\log_m n)$ 。

对于位置转成文档号这个问题,不仅可以利用 cache line 这一特性,还可以利用更多的其他特性,如文档的平均长度。假设每个文档的长度相差不大,文档的平均长为  $k$ ,每个 cache line 的大小为  $2^m$  字节,文档边界数组中的每个元素需要  $2^n$  表示。那么可以知道一个 cache line 平均能容纳  $k * 2^{(mn)}$  个位置,称之为 cache line 位置空间 (CLPS; cache line position space)。如果全部的位置数量为  $L$ ,那么 CLPS 的数量

为  $L/(k * 2^{(mn)})$ 。因此,可以使用一个辅助结构来索引这些 CLPS 在文档边界数组中的开始位置。

如图 8-6 所示,假设文档平均文档长度为 8,cache line 的大小为 8 字节,边界数组中的每个元素需要 2 字节,那么一个 cache line 平均可以包含的位置数是 32,总共有 99 个位置,所以需要 4 个 CLPS,每个 CLPS 指示这在空间在文档边界数组中的开始位置。例如,第二个 CLPS 从 32 到 63,而在文档边界数组中,边界为 40 的文档是第一个属于这个空间的值,那么在索引 CLPS 在文档边界数组中的开始位置时,就指向 4 这个位置。索引数组可以很快的建立,只需顺序扫描一遍文档边界数组,所以其时间复杂性为  $O(N)$ ,  $N$  是索引数据集中包含的文档。

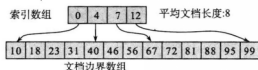


图 8-6 CLPS 结构示意图

利用索引数组,可以很快地把位置信息转成文档号,其过程见算法 8-3。

#### 算法 8-3 位置转成文档号

Step 1. 把位置值与 CLPS 的大小相除,得到其属于的 CLPS 的位置,并从索引数组中读取这个 CLPS 在文档边界数组的开始位置。

Step 2. 从得到的位置顺序扫描文档边界数组,直到遇到比这个位置大的边界号,返回下标值,其下标就是这个位置属于的文档号。

在这个算法中,有两个地方会引起缓存缺失,第一个地方是在 Step 1 中读取 CLPS 在文档边界数组中的开始位置,会引起一次 cache line 的缺失;第二个地方是顺序扫描文档边界数组时,平均会引起一次缓存缺失。因为 CLPS 所包含的位置数已都在一个 cache line 中,连续的访问不会导致缓存的缺失,其访问代价非常小。所以把位置转成文档号的时间复杂性为  $O(1)$ ,平均每次访问有 2 个缓存失效。

例如在图 8-6 中,我们要把第 45 号位置转成其对应的文档号。首先,我们得到这个位置属于第二个 CLPS(45 除 32),然后,通过访问索引数组中下标为 2 元素得出这个 CLPS 在文档边界数组中从第四个位置开始。最后,在文档边界数组中,从第四个元素开始顺序扫描,找第一个比 45 大的元素。在这个例子中,第一个得到的元素是 40,则跳过,下一个元素是 46,其值比 45 大,那么位置 45 属于这个文档,而其对应的下标是 5,所以位置 45 对应的文档号是 5。

### 第三节 倒排索引压缩

压缩技术是解决网络传输负担的有效技术。数据压缩有无损压缩和有损压缩两

种。在搜索引擎中用到的压缩技术属于无损压缩。接下来,我们先讲解各种倒排索引压缩算法,然后来分析搜索引擎索引中词典和倒排表的压缩。

## 一、倒排索引压缩技术

为了减少传输数据的时间,搜索引擎通常对倒排链进行压缩。文献(Scholer et al. 2002, Büttcher et al. 2007, Yan et al. 2009, Ding et al. 2011)表明,倒排索引的压缩不仅可以减少传输数据的时间,而且还可以提升查询处理的速度。在一般情况下,压缩算法对数值小序列能得到较高的压缩率,所以在压缩倒排索引之前,通常会对倒排链中一些递增序列(如文档号序列,文档中的位置信息序列)做  $d$ -gap 操作,使得要压缩的数据尽可能的小。 $d$ -gap 操作是对一组递增的序列进行转换,使其变成一组数值较少的序列。主要目的是减少需要编码数的数值大小,从而提高压缩率。对于一个递增的正整数序列, $d$ -gap 操作对每个数字使用  $d_i = d_i - d_{i-1} - 1$  ( $i > 1, d_0 = 0$ ) 进行替换。例如对 1, 3, 7, 20, 25, 60 这组文档号序列,其进行  $d$ -gap 操作后其转化为:0, 1, 3, 12, 4, 34。在还原时,只需要对数字进行  $d_i = d_i + d_{i-1} + 1$  ( $i > 1, d_0 = 0$ ) 操作。在倒排索引中,索引词对应的文档号通常是递增分配的,因此,我们可以使用  $d$ -gap 操作来减小文档号的值。另外,在词在文档中出现位置也是递增的,所以也可以使用  $d$ -gap 操作。

倒排索引的压缩算法按照数据压缩后的数据对齐方法,主要可以分成三类(Witten et al. 1999, Anh et al. 2001, Scholer et al. 2002, Trotman 2003, Anh et al. 2005, Héman 2005, Zhang et al. 2008, Yan et al. 2009, Yan et al. 2009, Anh et al. 2010):比特对齐(bit-aligned)、字节对齐(byte-aligned)和字对齐(word-aligned)的压缩算法。比特对齐的压缩算法对每个压缩的整数使用几个比特来表示,这系列代表算法如 Elias-gamma, Golomb, Rice 等。字节对齐的压缩算法对每个压缩整数用几个字节来表示,如 VariableByte。字对齐的压缩算法通常在一个字中(通常由 4 字节或 8 字节构成)中尽可能多压缩数字,如 Simple9。另外,还有一些对固定个数的压缩算法,对固定长度的数字进行压缩,例如每次压缩 4 个数字(Group Variable Byte)(Dean 2009),128 个数字(PForDelta)(Héman 2005; Zukowski et al. 2006, Zhang et al. 2008, Yan et al. 2009),另外文献(Witten et al. 1999, Yan et al. 2009)把压缩算法分成全局模型和局部模型两类。在全局模型中,使用统一的模型对各个倒排链进行压缩,例如 Gamma, Delta, Variable Byte 等。在局部模型中,每个倒排链可以根据需要对压缩算法中的参数进行调整,例如 Rice。另外,全局模型的编码方法可以有参数的,也可以是无参数的。局部模型通常是有参的,其压缩率往往好于全局模型,但其解压速度往往慢于全局模型。本文对解压算法的分类主要使用对齐方法进行分类。

### 1. 比特对齐的压缩算法

比特对齐编码方法主要思想是对一个整数用多个比特来编码。也就是说这些

压缩算法对一个数据压缩,可以压缩成任何长度的比特。以下是一些比较常用的比特对齐的编码:

Unary 编码对一个正整数  $n > 0$  通常编码成  $n-1$  个为 0 的比特与 1 个为 1 比特的值。例如对于 5, 其编码为 00001。Unary 编码是一种信息熵编码,它对于符合  $p(n) = 2^{-n}$  分布的数字的压缩效比较好。另外 Unary 编码也是一种前缀编码,所以可以自解压的。基于以上特性,Unary 经常与编码结合,以达到较高的压缩率。

Binary 编码是在计算机中经常使用的编码方式。对于  $0 \leq k \leq K$ ,  $k$  可以使用  $\lceil \log_2 K \rceil$  比特进行二进制编码。例如在计算机中一个整数通常使用 32 比特进行编码。但是 Binary 编码不是前缀编码,只有知道具体编码的比特数才能正确的解码数据。文献(Moffat and Anh 2006)提出了两种不同的 Binary 编码, RBUC 和 BASC。在 RBUC 编码中,首先对长度为  $m$  序列按每组大小  $s$  分成  $\lceil m/s \rceil$  组,其次为每组计算一个  $b = \lceil \log_2(1+x) \rceil$ ,  $x$  是当前组中最大值。然后对组中的数用  $b$  比特进行编码。最后递归的对每组产生了  $b$  按上述方法进行编码,直到剩余需编码数为 1。在 RBUC 中,组大小  $s$  的选择非常重要,例如可以选  $f_1(s) = s$ ,  $f_2(s) = 2s$  等,其对应需要递归编码的次数分别是  $\log_2 m$  和  $\sqrt{\log_2 m}$ 。BASC 是一种在线(On-line)的编码方式。在线编码方式指的是不需要知道当前编码数之后的数就可以对当前数进行编码。BASC 使用  $b_{i-1}$  来预测序列中下一个值  $x_i$  需要的比特数  $b_i$ 。在这种编码中,如果  $b_i \leq b_{i-1}$ , 则输出 1 再使用  $b_{i-1}$  比特对  $x_i$  进行编码,否则先使用 Unary 对  $b_{i-1} - b_i$  进行编码,再使用  $b_i - 1$  位对  $x_i$  低  $b_i - 1$  位进行编码。BASC 的一些变种算法使用更多的信息来预测  $b_i$  从而避免一些异常数导致预测  $b_i$  突然变化太大,例如取前  $m'$  个数的  $b$  的平均数来预测  $b_i$ , 或者限制  $b_i$  每次最多变化 1。

Elias 编码(Elias 1975)是一种非参编码(非参数编码指使用静态或固定长度的编码来编码一个正整数)。Elias Gammas 编码对于一个正整数  $k$ , 其首先把  $k$  拆分成两部分,第一部分为  $n$  ( $2^n \leq k < 2^{n+1}$ ), 其使用 Unary 编码进行编码;第二部分是  $k \leq 2^n$ , 这一部分使用长度为  $n$  比特的二进制编码。例如整数 5 编码的结果是 0101( $5 = 2^2 + 1$ )。另外,对于整数 1, 其直接编码成 1。显然,这种编码方法对于数值比较小的整形数压缩的效果比较好。但对于数值较大的值( $k > 15$ ), 其编码方法并不高效(Williams et al. 1999), 原因是使用 Unary 对较大的  $n$  进行编码需要更多的存储空间。Elias Delta 编码方式与 Elias Gammas 编码相似,不同点是对于第一部分的  $n$  用  $n' = n + 1$  表示,并对  $n'$  进行 Elias gammas 编码。这种方式的编码对于大数比较好,但对于数值比较小的值,编码方法没有 Elias gammas 的效果好。GammasDiff(Yan et al. 2009)也是对 Gammas 编码另一种改进。它首先对需要压缩的一组数计算出一个平均数(假设需要  $n'_1$  比特表示),然后用  $n - n'_1$  代替  $n$  对第一部分进行编码。这种编码方法对于压缩数与的平均数相差不大的序列进行编码效果比较好。

Golomb 编码(Golomb 1966)是一种有参数的编码方法,需要给定参数  $b$  来正

整数  $k$  进行编码。在编码时,仍将编码拆分成两部分:第一部分是商:  $q = \lceil (k-1)/b \rceil + 1$ , 通常使用 Unary 编码对  $q$  进行编码。第二部分是余数:  $r = k - q * b + 1$ , 使用紧凑二进制编码(Truncated binary code)来编码。其编码方法如下:设  $m = \lceil \log_2 b \rceil$ , 如果  $0 \leq r < 2^m - b$ , 则对  $r$  使用  $m-1$  比特长度进行编码;如果  $2^m - b \leq r < b$ , 则使用  $b$  比特长度进行编码。例如取  $b=3$ ,  $r$  的取值只有三种情况:  $r=0$ ,  $r=1$  及  $r=2$ , 因此对  $r$  可编码成 0, 10, 11。对于参数  $b=3$  的 Golomb 编码对整数 5 编码时:  $q=1$  使用 Unary 编码成 1,  $r=2$  编码成 11, 因此整个编码为 111。在 Golomb 编码中, 参数  $b$  的选择非常关键, 直接影响其压缩率与解压时间。文献(Witten et al. 1999)给出了一种  $b$  取值的方法:  $b = 0.69 * \text{mean}(A)$ , 其中  $A$  是一组需要编码的数。对于一个倒排链, 可以通过  $k = N/f_i$  计算出文档号的均值, 其中  $N$  是文档的总数,  $f_i$  是倒排链的长度。

Rice 编码(Rice et al. 1971)是 Golomb 编码的一个变种。它限定  $b$  只能取 2 幂数。好处是对于余数  $r$ , 只有一种长度编码方式(一般使用二进制编码), 不需要额外计算编码长度。另外, 在计算机中, 可以使用移位操作来达到快速编码。缺点是  $b$  被限制成 2 幂数, 在一些情况下其编码的压缩率比 Golomb 编码低。RiceV<sub>T</sub>(Moffat et al. 1992, Witten et al. 1999, Yan et al. 2009)是 Rice 编码的变种, 主要是对  $b$  选择做了一些限制。

LLRUN 编码(Fraenkel et al. 1985)和 Elias Gammas 编码相似, 其不同点是对第一部分  $n$  ( $2^n \leq k < 2^{n+1}$ ) 的编码: 在 Elias Gammas 编码中使用 Unary 编码进行编码, 而在 LLRUN 编码中使用 Huffman 编码进行编码。Huffman 编码可以基于整个倒排索引统计也可以对每个倒排链统计。这种编码方式对倒排索引中位置信息的编码能带来较大的压缩率。

插值编码(Interpolative Coding, IPC)(Moffat et al. 1996, Moffat et al. 2000)与其他编码方式不同, 它是直接对原始递增序列进行编码, 而不是经  $d$ -gap 操作后的数。对于文档号为  $l < d_i < d_{i+1} < \dots < d_j < r$  的序列, 其中  $l, r$  是在编码时已知的序列的最小值与最大值。它首先对  $d_m$  进行编码,  $m = (i+j)/2$ , 然后分别对  $d_i \dots d_{m-1}$  和  $d_{m+1} \dots d_j$  两个序列的数进行递归编码。这两个序列的最小值与最大值分别为  $(l, d_m)$  和  $(d_m, r)$ 。对  $d_m$  解码时, 可以推断出  $d_m > l + m - i$  且  $d_m < r - j + m$  (因为在  $d_m$  左边有  $m - i$  个数,  $d_m$  的右边有  $j - m$  个数), 因此, 可以确定  $d_m$  的编码范围  $[0, x]$ ,  $x = r - l - j + i - 2$ , 其编码所需的比特数是  $\lceil \log_2(x+1) \rceil$ 。另外, 在还原  $d_m$  时, 还需要加上  $l + m - i + 1$ 。文献(Yan et al. 2009)对 IPC 编码进行了优化, 主要使用紧凑二进制编码对  $d_m$  以节省更多的空间。另外, IPC 可以对整个倒排链进行一次编码, 也可以在一个块中的数据进行编码。通常对块进行编码时, 选取  $2^b - 1$  个数成一块能达到较高的压缩率, 如 127 个文档号组成一个块。

## 2. 字节对齐的压缩算法

在计算机中, 通常是按字节为单位进行数据处理。因此比特对齐的压缩方式在

编码与解码时,往往会消耗更多的 CPU 处理时间,这直接影响编码与解码的速度。字节对齐压缩是对一个整数使用一个或多个字节来编码。字节对齐的压缩算法主要有 Variable Byte 编码。

Variable Byte 编码(Williams et al. 1999)对正整数  $k$  编码成一个字节的序列。在每个字节中的第一个比特通常用于表示下一个字节是否属于当前整数  $k$  的编码(例如 1 表示属于,0 表示不属于),其他 7 个比特以二进制的方式表示部分  $k$  中的内容。例如对 5,128 这两个数使用 Variable Byte 编码,其分别需要 1 字节与 2 字节来表示编码结果:00000101,10000000 00000001。在 Variable Byte 编码中,编码一个数字至少需要 1 个字节,所以其压缩率上没有比特对齐的压缩方法好,但其解压速度往往比比特对齐的压缩算法快(Williams et al. 1999, Witten et al. 1999, Scholer et al. 2002, Trotman 2003, Zobel et al. 2006, Zukowski et al. 2006)。

### 3. 字节对齐的压缩算法

字节对齐的压缩算法(Anh et al. 2004, Anh et al. 2005, Anh et al. 2006, Anh et al. 2006, Zhang et al. 2008)通常以计算机中的字(由 4 字节或 8 字节组成)作为数据压缩后的存储单位,尽量在每个字中编码多个数字。对于需要压缩数字,这些压缩算法通常以最小二进制编码对压缩数进行编码。由于二进制编码不是前缀编码,不可自解压,所以如何区分这些混合存储的二进制编码的数字是这系列算法的关键。这些算法一般使用字的前几个比特来存储这个字中数据的编码方式。通常,这些压缩算法自身保存着一张数据解压的映射表, key 是字中存储编码方式的数值, value 说明字的数据部分编码了多少个数字,以及各个数字占用了多少比特。根据使用字大小的不同可分成 Simple 和 SimpleX64,分别对应使用 4 字节与 8 字节来存储数据。通常 Simple 系列是在长度为 4 字节的字中取前 4 比特来表示编码方式,后 28 比特来存放编码后数据。SimpleX64 是在前  $n(3 < n < 9)$  比特中来存放编码方式,后  $64-n$  比特来存放数字,这是  $n$  取决于具体的应用。

Simple9(Anh et al. 2005, Anh et al. 2006, Anh et al. 2010)共使用 9 种不同的方式来区分 28 比特数据中的数字:28 个 1 比特数字,14 个 2 比特数字,9 个 3 比特数字,7 个 4 比特数字,5 个 5 比特数字,4 个 7 比特数字,3 个 8 比特数字,2 个 14 比特数字,1 个 28 比特数字。在解码时,只需要读取前 4 个比特的值,通过查找解压表,就可以知道字中后 28 比特数字的编码方式及总共编码了多少数字。这种编码的主要缺点是对于大于 28 比特的数字无法进行编码。另外在一些情况下,28 比特中的一些位没有被使用到,如在 5 个 5 比特数中就有 3 个比特没有被使用,所以其存储空间上的利用不是很紧凑。

Simple16(Zhang et al. 2008)是对 Simple9 的改进。Simple 使用 4 比特来表示编码方式,而在 Simple9 中只使用了 9 种编码方式,所以有 7 种编码方式被浪费了。在 Simple16 定义了另外 7 种不同的编码方式来使用 Simple9 中一些编码方式



中没有使用的数据位。例如在 Simple9 中存 5 个 5 比特数字, 这里有 3 个比特的数据位没有被使用。在 Simple16 中, 其增加了 3 个 6 比特与 2 个 5 比特的编码方式。Simple16 虽然比 Simple9 的编码方式更多, 利用的存储空间更紧凑, 但其还是没有解决大数压缩的问题。另外, Simple 系列的算法可以根据压缩数据的特点, 对编码方式进行优化。例如, 当压缩的数都是小数时, 可以针对小数多几种编码方式, 例如文献(Yan et al. 2009)中的 Simple16-128 就采用的类似的思想。

SimpleX64(Anh et al. 2010)是使用 64 比特字来进行编码。在 simple 的编码中, 因为只有 4 个比特来表示 16 种不同的编码方式, 但在现实的情况中, 这些编码方式还不够用。所在 simpleX64 中使用更多的比特来表示编码方式, 例如使用 6 个比特来表示编码, 后面 58 比特数据。与 Simple 编码一样, 其预先定义 64 种数据编码方式。在具体使用时, 可以根据需要来设置编码方式的比特数。通常, 在 64 位机器上, 这种压缩算法的压缩率及解压速度都略好于 Simple 系列算法。

#### 4. 其他压缩算法

PForDelta(Héman 2005, Zukowski et al. 2006, Zhang et al. 2008, Yan et al. 2009)是一种块数据压缩算法。它每次对  $m$  个正整数进行编码,  $m$  是 32 的整数倍, 例如  $m=128$ 。在编码时, 它首先计算参数  $b$  使得大部分压缩数字(例如 90%)小于  $2^b$ (对于大于  $2^b$  数字, 它称为异常数字, 小于  $2^b$  称为正常数字)。其次分配  $m$  个  $b$  比特槽来存放正常数字和异常数出现位置。每个正常数字按二进制编码存放在对应的槽中, 例如第 5 个数字是正常数, 那么存放在第 5 个槽中。对于异常数对应的槽, 它存放下一个异常数出现的位置。例如当前位置为  $i$  槽中存储的值为  $x$ , 则下一个异常数对应的位置是  $i+x+1$ 。对于两异常距离大小  $2^b$  时, 强制设置  $i+2^b$  为异常数。异常数则按其出现的顺序存放在  $m$  槽后, 根据最大异常数值的大小使用 8 比特, 16 比特, 32 比特来存储(Zhang et al. 2008)。对于第一个异常数的位置, 参数  $m, b$  值和异常数的范围存放在头部元数据中, 这个元数据可以使用 2 或 4 字节进行表示。在解码时, 只需要两步就可以对  $m$  个数进行解码。第一步是根据  $b$ , 把  $m$  个槽中数据拷贝到解码后的整形数组中。第二步, 根据异常数的出现位置把异常数拷贝到对应的位置。这种解码方式对现代的计算机体系结构可以做得非常快, 第一步中只涉及数据的拷贝, 不需要进行任何条件判断与分支预测, 第二步相对较慢, 但异常数的个数往往比较少。这种算法的解压速度非常快, 而且压缩率比字节对齐的高。

PForDeltaOpt 编码(Yan et al. 2009)是对 PForDelta 在压缩率上的一种改进算法, 主要体现在异常数的处理上。在传统的 PForDelta 压缩算法中, 如果两个异常数相隔大于  $2^b$ , 需要在中间多加入异常数, 这会导致异常数增多, 从而导致压缩率变差, 解压速度变慢。这种情况在压缩数据较小的序列很常见, 例如两个异常数相差 40,  $b=2$  时, 需要强制使两异常数中间的 10 数为异常数。这种情况下不能简单的增

大  $b$ , 因为增大  $b$  会使得大部分正常数的编码空间开销增加。PForDeltaOpt 中一个改进是使用异常数数组来保存其出现的位置和数值。在这种方法中, 异常数对应的槽中存放异常数的低  $b$  个比特数据, 其余部分的值存在异常数数组中。对于这个异常数数组, 可以通过 Simple16 来得到较大的压缩率。另一个改进是对异常数的比例进行优化, 例如让异常数的比例浮动在  $[0, 0.5]$  之间, 然后选取一个压缩率最高的  $b$  进行压缩。这个优化方法有可能产生的异常数比较多, 因而会影响解压的速度, 但其压缩率会提升。

Group Variable Byte 编码(Dean 2009)是对 Variable Byte 编码在解压速度上的一种改进。虽然 Variable Byte 编码解压速度比特对齐压缩方法的解压速度快, 但是在解码的过程仍包含了大量的条件判断、移位和掩码操作, 直接影响了解压的速度。一种改进的方式是用 2 比特来表示编码后的长度, 来减少解码过程中使用到的操作。例如对 2 比特对 5, 128 进行编码: 00000101, 01000000 00000010。这种编码方法的主要问题是只能对小于 30 比特数字进行编码, 并且在解码的过程中仍需要一些移位操作。Group Variable Byte 编码每次对 4 个正整数进行编码, 得到长度为 5~17 字节序列。在序列中的第一个字节用于保存这 4 个数字的编码方式: 每 2 比特存放其对应数字编码后的长度, 例如 00 表明其编码长度为 1 字节。在解码时, 可以通过查找解码表来提升解压速度。解码表的大小为 256 项, 每个项包含编码数字在序列中的偏移和解压此数字的对应掩码。例如, 对于 00000110 其对应解码表中的第 6 项数字, 其对应的内容是: 偏移 {1, 2, 3, 5}, 掩码 {ff, ff, ffff, ffffff}。这些编码方式只需要掩码操作就可以很快的解码, 所以其解码速度比 Variable Byte 快。在一些情况下, 这种编码比 Variable Byte 压缩率更差。例如需要编码是数小于 128, 则 Group Variable Byte 比 Variable Byte 压缩每个数多开销 25% 的存储空间。

## 5. 序列转换

对一个递增的序列, 可以通过  $d$ -gap 操作使这个递增的序列转化成个值较小的序列, 从而提升对这个序列的压缩率。在这个小节中, 我们主要介绍其他的序列转换方法, 这些序列的转换方法的目标也是为了把目标编码序列转化成值较小的序列, 从而提升序列的压缩率。

Burrows-Wheeler Transform(BWT): BTW 转换(Burrows et al. 1994)主要思想是对序列经过轮转后得到一个序列矩阵。然后在这个矩阵上做排序与变换操作, 使得把原始序列中相同数字尽可能的聚在一起。它往往与 MTF 和 RLE(Run-Length-Coding(Moffat et al. 1992))一起使用, 使得序列值变小。

Move-To-Front(MTF): MTF 序列转换(Bentley et al. 1986)的主要思想是对需编码数字用其在另一数组中的索引值来代替。通常这个索引值小于需要编码的数字。例如对数值范围在  $[1, 4]$  之间的序列  $\langle 4, 4, 1, 4, 2 \rangle$  进行编码时, 其编码的过程如表 8-1 所示, 经过 MTF 的序列转换后, 其序列变成  $\langle 4, 1, 2, 2, 3 \rangle$ , 在新的序列

中相对于原序列的值要小。在序列还原时,经过同样的操作就可以得到原始序列。另外一些对 MTF 的优化方法(Yan et al. 2009)主要对编码值是否移到索引序列的最前端,有些方法只移到  $i/2, 2 * i/3$  的位置( $i$  是当前编码值在索引序列中的位置)。

表 8-1 MTF 对序列<4,4,1,4,2>进行转换的过程

编码的值	MTF 产生的序列	索引序列
4	4	<1, 2, 3, 4>
4	1	<4, 1, 2, 3>
1	2	<4, 1, 2, 3>
4	2	<1, 4, 2, 3>
2	3	<4, 1, 2, 3>

Most-Likely-Next(MLN)(Yan et al. 2009):MLN 使用一张映射表来对编码序列进行转化。在映射表中,存储着每个数字后面最可能出现的数字。例如,我们把使用映射表转换的数字大小限制在 10 以内,则存储映射表需要  $10 * 10$  大小的数组。使用 MLN 对序列进行转换时,使用该值在映射表中位置进行替换。对于大于 10 的值不进行替换,直接使用原始值。虽然 MLN 需要为每个存储一张解压的映射表,会增加存储开销,但通过这张映射表,可以加快数据还原的速度。

## 二、词典与倒排表的压缩

### 1. 倒排索引中字典压缩

字典大小的估算:通常,可以使用 Heaps 定律(Heaps 1978)来估算,一个数据集字典的大小  $M = k * T^b$ ,  $k$  是在整个文档集中词的数量(包含相同的词),通常参数  $k, b$  取值:  $30 < k < 100, b = 0.5$ 。从这个公式中可以得出:字典的大小随着数据集的增大而增大。所以对于一个大的数据集,尤其是网页数据集,其对应的字典往往很大。例如对于包含 2500 万网页 GOV<sub>2</sub> 数据集,其词典大小将近 5000 万。为了加快查询处理速度,字典往往被检索系统加载到内存中。因此字典压缩在倒排索引的结构中是一个关键的问题之一,以下是常用的几种词典编码(Witten et al. 1999)。

定长串编码:一种简单的字典组织方式就是对索引词按字典顺序排序,然后为每个索引词分配一个固定长度的值,如 20 字节。在查找时,通过二分查找找出查询词的相关信息。这种方法主要缺点是需要的存储在空间很大。在英文数据集中,通常一个索引词的平均大小为 8 字节,如果按固定长度 20 字符存储,其浪费近 60% 的空间。另外,对于长度大于 20 字节的词需要进行截断。好处是因为每个索引词长度都

是固定的,所以可以随机的访问。

**变长串编码:**变长串的压缩的主要思想是对所有的索引词的字符串放到一个大的字符数组中。然后用索引词在字符数组中的偏移来替换索引词。在查找时,通过这个偏移就可以从字符数组中得到对应的索引词的内容(其长度是下一个索引词的偏移与当前索引词偏移之差)。这种方式的优点是字符串的存储开销比定长串的低,只增加了一个偏移地址(通常是4字节)。但是,这种编码方法在查询中,需要二次随机访问才能得到对应索引词的内容(第一次通过字典下标获取在字符数组中的偏移,第二次从字符数组中获取索引词的内容)。

**块编码:**如果字典存放在磁盘中,那么定长串编码和变长串编码对字典的访问效率非常低。一种解决方案就是对字典进行块编码。首先对字典中的词按字典序排序,并分成 $N$ 组,对每一组选取块中第一个索引词的内容和这个块在字典中偏移放入块索引数组中。块索引数组往往比原先的字典小很多,可以加载到内存。在查找时,先通过二分查找在块索引数组中找到对应块偏移地址,然后在块中按顺序查找对应词的信息。这种方法在字典比较大,且内存受限的情况下使用比较高效。

**前端编码:**前端编码的主要思想是对于一个有序的字典中,相邻两个索引词往往有共同的前缀。在编码时,可以对前缀信息进行编码以达到字符串压缩。在编码中,每个字符中都存取2个整数。第一个表中与上一个字符串共同的前缀的长度;第二个表示去掉前缀后还剩字符串的长度。这种编码可以与块编码进行结合,以达到较高的词典压缩率。

**哈希编码:**对于一个静态的字典,可以通过最小完美哈希编码(Witten et al. 1999)来对字典进行编码。如果字典的大小为 $M$ ,那么可以通过最小完美哈希把这些字符串无冲突映射到 $[1, M]$ 中。但对于动态的字典,可以通过一些哈希函数给字典中的每个索引词计算哈希值,然后在词典中只保存哈希值。通常为了减少冲突的可能性,这个哈希值是64位。这种方法的优点就是每个词的长度都是固定的(8字节),而且可以在 $O(1)$ 的时间内找到对应词的信息。但是这种方法的缺点就是无法进行通配符搜索。例如: `text *`, `te? t`等。另外,还需要处理哈希值冲突的情况。

表8-2中(Witten et al. 1999)对包含100万词条的词典使用不同编码所需要的空间的统计,我们可以看出定长串编码所需要的空间最大,而哈希编码的空间最小,其次是前端编码。当前的搜索引擎中通常采用前端编码(Hatcher et al. 2005)或哈希编码对组织词典。原因是搜索引擎索引上千万的网页,其产生的词典是非常巨大的,例如GOV2有5000万的词条。前端编码可以较好的压缩词典,哈希编码在压缩词典时还可以高速的访问词典。

表 8-2 对包含 100 万词条的词典使用不同编码所需要的空间

字典编码方法	字典大小/MB
定长串编码	28
变长串编码	20
块编码	18
前端编码	15.5
哈希编码	13

## 2. 倒排表的压缩

在一个索引词的倒排链中,主要存放着出现这个词的文档号序列,词在文档中的频率及词在文档中出现的位置信息。这三种序列数值大小和分布是不一样的,另外这些信息的访问频率也不太一样。例如文档号序列访问的比较频繁,位置信息访问的次数相对较少。原因是在一般的查询处理过程中(Broder et al. 2003, Dean et al. 2004, Yan et al. 2009),通常使用文档号来进行布尔查询,过滤出符合要求的文档,然后再访问词频等信息对文档进行排序,最后访问词在文档中出现的位置信息,对文档进行重排序。因此,在倒排表的压缩中,我们需要分别对这三种数据的压缩进行讨论。

### (1) 文档号序列的压缩

一个索引词对应的倒排链中的文档号是递增分配,为了增加文档序列的压缩率,我们会对这个序列进行  $d$ -gap 操作。通过这个操作,使得一个有序的序列变成了一组值较小的序列。而大部分的压缩算法对较小数值的压缩率非常好。文献(Blandford et al. 2002, Shieh 2003, Silvestri et al. 2004, Silvestri et al. 2004, Blanco et al. 2005, Blanco et al. 2006, Cheng et al. 2006, Silvestri 2007, Yan et al. 2009; Ding et al. 2010, Lavee et al. 2011)研究如何分配文档号,使得经过  $d$ -gap 操作后,能得到更多数值较小值,从而得到更小的索引。

文档号重分配:文档重分配问题是一个最优化求解问题,其目标是文档号经过置换后,在特定压缩算法下得到的索引大小最小。这个问题是一个 NP 难问题(Blandford et al. 2002, Shieh 2003),有多种算法可以近似的求解这个问题。这些近似求解的算法的主要思想是对相似的两个文档尽量分配相邻的文档号。这里的文档相似通常定义成两文档共现的词的数量。

文档重分配的工作主要可以分成三大类:在第一类工作中,其定义图  $G = \{D, E\}$ ,  $D$  是所有文档的集合,  $E$  是图中边的集合,每条边  $\langle d_i, d_j \rangle$  的权重是两文档  $(d_i, d_j \in D)$  的相似度。文献(Shieh 2003)把文档重分配转化成旅行商问题

(TSP),即遍历图中每个节点一次,使其走过的路径和最大,并使用贪心最近邻算法(GNN; Greed Nearest Neighbors, GNA; Greed Nearest Addition)和最大生成树算法(MaxST; Maximum Spanning Tree)对这个问题进行求解。作者发现 GNN 的效果是最好的,但这系列的算法复杂度太高,对大规模文档集进行文档号重分配变得很困难。为了减少计算复杂性,文献(Blanco et al. 2005)使用 SVD(Singular Value Decomposition)来减少词与文档号矩阵的维度。文献(Ding et al. 2010)使用 local sensitive hashing 来减少图  $G$  中的边数,得到新的图  $G'$ ,然后在  $G'$  用近似的 TSP 算法进行求解。这样做的好处是可以对大规模的文档集进行文档号重分配。第二类工作是使用聚类的思想,首先把相似的文档聚成一类,然后为每个类中分配相邻的文档号。文献(Blandford et al. 2002)使用自上而下的方法对文档集递归的分成多个类,然后为类中的相似文档分配相邻的文档号。文献(Silvestri et al. 2004, Silvestri 2007)提出 k-scan 方法对文档集自底而上的进行聚类。文献(Blanco et al. 2006)结合 k-scan 方法和 TSP 方法,首先使用 k-scan 得到相似文档的类,然后在类中使用 TSP 方法进行文档号的分配。这些工作的算法复杂仍很高,不利于大规模文档集文档号的重分配。第三类工作是使用 URL 序来分配文档号(Silvestri 2007)。这种方法首先对文档按 URL 进行排序,然后按 URL 序对文档进行重分配。这种文档号重分配方法比较简单,可以用于大规模文档集的文档号重分配工作,而且效果也非常好,但只适用于 Web 文档集的重分配工作。另外,文献(Lavee et al. 2011)提出一些文档分配方法来解决有多台机器存在的情况下,如何有效地进行文档号的重分配问题。

图 8-7(Yan et al. 2009)展示了在 TREC GOV2 数据集上,文档号按原始序(original),随机序(random)和 URL 序(sorted)进行分配后,倒排链的  $d$ -gaps 分布图。 $d$ -gaps 的统计使用了 2171 个不同查询词的倒排链(这些查询词包含于 GOV2 提供的查询集中随机选取 1000 个查询)。通过图 8-7 可以发现:使用文档重排序后得到较小的  $d$ -gaps 数字明显多于其他的文档号分配方法。图 8-8(Yan et al. 2009)展示了不同的压缩算法在不同的文档号分配策略下,每个查询对应文档号序列平均需要的存储空间。在图中可以发现:①文档重排序后,大部分压缩算法相对于其他分配策略都可以提升文档号序列的压缩率。②所有的压缩算法在文档号按原始序分配比随机序分配有一点提升,但效果并不明显。③IPC(使用块压缩的方法)压缩算法的压缩率是最高,但文献(Yan et al. 2009, Yan et al. 2009)表明其解压的速度相对于其他压缩算法要慢很多。④OptPFD(PForDeltaOpt)算法的压缩率与其他比特对齐的压缩算法具体可比性。在文档按 URL 序进行重分配后,OptPFD 的压缩率比字节对齐的压缩算法(var-byte)和字对齐的压缩算法(Simple9, Simple16, Simple16\_128)高。图 8-9 是各种压缩算法在文档号的解压速度,从图中可以看出 OptPFD 是解压速度是最快的,而 IPC 的解压速度最慢。

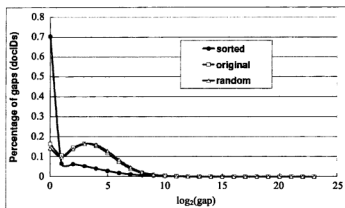
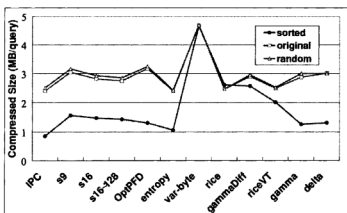
图 8-7 倒排链中文档号之间的  $d$ -gaps 分布图

图 8-8 不同文档号分配下平均每个查询对应文档号序列的压缩大小

## (2) 词频序列的压缩

倒排链中的词频序列的数值是非递增的,而且通常是一些数值较小的值,对于这一块数据的压缩也比较关键。首先分析一下文档重分配与词频链的影响。文档重分配后,每个词在各个文档中的词频信息是不会改变的,因此整个倒排链的词频分布也不会改变。但是文档重分配后,可以把类似大小的词频信息聚到一起,所以对数据顺序相关的压缩算法(如 Simple16)影响较大,而对与数据顺序无关的压缩算法(如 rice)基本没有影响。图 8-10(Yan et al. 2009)展示了文档号在不同分配策略下,使用不同的压缩算法对词频链进行压缩后,平均每个查询词频链占用的空间大小。从图中可以得出,与压缩数据顺序相关的算法(entropy 左边)可以从文档重分配中得到较大的好处,而与压缩数据顺序无关的算法(entropy 右边)无论怎么分配文档

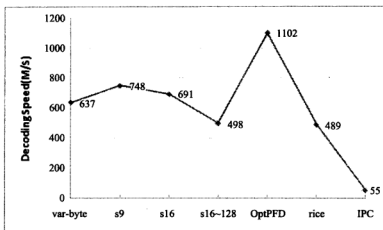


图 8-9 不同压缩算法对文档号的解压速度

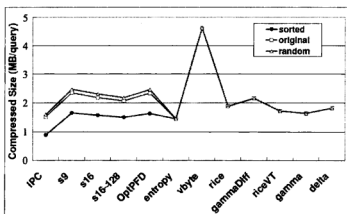


图 8-10 不同文档号分配下平均每个查询对应词频序列的压缩大小

号,对其没有影响。在图中 IPC 的压缩率仍是最高。

其次讨论如何使用 MTF、MLN 序列转换方法对词频序列进行转换,使得词频链的压缩更高效。表 8-3(Yan et al. 2009)是使用 MTF 和 MLN 两种方式对词频链进行转换后,使用不同的压缩算法得到平均每个查询需要的存储空间大小。从表中可以得到:①MTF 和 MLN 对词频链的转换可以提升其压缩率(除 var-byte)。②entropy 的方法经序列转换后,能得到更大的好处。③MLN 的转换通常比 MTF 的转换好,虽然需要在倒排链的开始多存储一个映射表。



表 8-3 平均每个查询对应词频链的空间大小(文档号按 URL 序分配)

	Basic	MTF	MLN
Block IPC	1.21	0.89	0.89
S9	1.65	1.53	1.52
Sl6	1.57	1.44	1.43
Sl6~128	1.50	1.38	1.37
OptPFD	1.63	1.43	1.31
entropy	1.45	1.13	1.14
var-byte	4.63	4.63	4.63
Rice	1.88	1.70	1.69
gammaDiff	2.16	1.80	1.79
RiceVT	1.72	1.44	1.43
Gamma	1.64	1.52	1.28

图 8-11 (Yan et al. 2009) 是不同压缩算法解压词频的速度。从图中看出各个压缩算法在词频链上的解压速度与文档号序列相差不大。词频序列经过 MLN 转换后, 影响解压速度比较明显。因为这时还需要使用 MLN 的映射表来还原原始值。

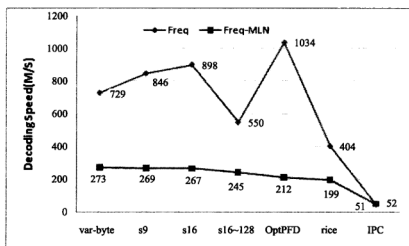


图 8-11 不同压缩算法对词频的解压速度

### (3) 位置序列的压缩

研究表明 (Moffat et al. 2000), 一个词在文档中出现具体聚集性 (一个词出现后有很大的概率会接着出现), 而不是随机出现在文档的各个地方。文献 (Bookstein et al. 1994, Moffat et al. 1996, Bookstein et al. 1997, Moffat et al.

2000, Moffat et al. 2006)基于这个特性,提出一些压缩算法来高效的对位置序列进行压缩。通常这些方法需要文档足够的长(如书本),来挖掘出位置信息的聚集性。对于一个网页文本,其长度往往很短,而且在倒排索引中两个相邻的文档可以没有任何关系,所以挖掘词在文档中出现的聚集性就变得非常困难。传统对倒排链的位置信息的压缩主要假定其均匀出现的文档中,但是词在网页文档中的聚集性还是存在的。因此,如何利用这个特性来提升位置序列的压缩率是一个比较关键的问题。因为在倒排索引中,位置序列的长度是最长的,一般是文档号序列长度的2~5倍。另外,对位置信息,在查询处理时,可能只需要少量的访问几篇文档的信息。

文献(Yan et al. 2009)对网页文本中位置压缩的研究表明,使用页面自适应(Page-Adaptive)的压缩算法能够得到较高的压缩率。在页面自适应的算法中,利用词在文档中的出现次数(词频)和文档自身的长度这两种信息来调整压缩算法。这两种信息在压缩位置信息时是已知的。例如,Page-Adaptive Rice Coding(PA-RC)中对Rice算法的 $b$ 的估算可以采用 $b = |d| / (f_{i,d} + 1)$ ,  $|d|$ 是文档的长度, $f_{i,d}$ 是词 $t$ 在文档中的词频。Remaining Page-Adaptive Rice Coding(RPA-RC)使用的是剩余的文本长度与词在剩余文本中的词频来估算 $b$ 。对RPA-RC算法还可以根据先前的 $b_1$ 和剩余文本的 $b_2$ 平滑的估算 $b$ ,思想类似于BASC编码(Moffat et al. 2006)中的平滑方式。

图8-12(Yan et al. 2009)是在GOV2数据集上,平均每个查询采用不同压缩算法对应的位置信息需要的存储空间。图中的list-Rice和list-RiceVT是使用整个序列来估算参数,list-LLRUN为每个倒排链的位置信息建Huffman表,RBUC和BASC是文献(Moffat et al. 2006)两种基本方法,在RBUC中 $f(s) = s * s$ ,BASC是没有经过平滑的。Page-IPC(Moffat et al. 1996, Witten et al. 1999)使用了

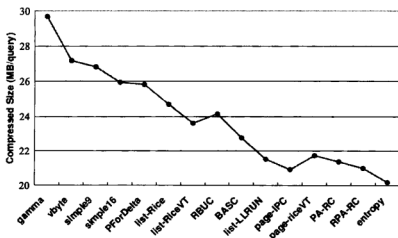


图 8-12 平均每个查询对应的位置信息需要的存储空间

页面的信息对 IPC 编码进行优化。从图 8-12 中,可以看出:①无参压缩算法(gamma, vbyte, simple9, simple16, PForDelta)或者使用整个序列的信息压缩算法(list-Rice, list-RiceVT)没有页面自适应的算法(page-IPC, page-riceVT, PA-RC, PRA-RC)好。②list-LLRUN 压缩算法要好于一些页面自适应的算法,但它在编码前需要统计出一张 Huffman 编码表,所以不是在线的编码方式,而页面自适应的算法基本上是在线的算法。③page-IPC 压缩算法能达到最高的压缩率,缺点是解压速度比较慢。

## 第四节 索引剪枝

搜索引擎通常使用倒排索引来高效的处理查询。因此,倒排索引的大小和查询词对应的倒排链的长度,直接影响着查询处理时所需要读取数据量及处理数据的时间。索引剪枝主要是从减少倒排索引的大小或者查询处理时尽量少的处理数据这两方面来提升查询的处理速度。按照索引剪枝发生阶段,把索引剪枝的方法划分为两类:静态索引剪枝方法和动态索引剪枝方法,其中动态索引剪枝方法在一些文献中经常被称为查询提前截断。

静态索引剪枝的方法主要发生在索引构建阶段,主要从减少倒排索引的大小和缩短倒排链长度的角度来提升查询的处理速度。通常的做法是在索引的构建阶段,预测出那些对未来查询处理不太有影响的信息,然后把把这些信息从倒排索引文件中去除。通过去除那些不重要的信息,倒排索引的大小和倒排链的长度可以缩短到非常小。在查询处理阶段,从剪枝完的索引中读取和处理的数据就会比使用完整的索引时少很多,从而提升查询的处理速度。这系列剪枝方法的关键问题是如何在索引的构建阶段有效地预测出哪些信息可以去除。因为被去除的信息在查询阶段是无法还原的,所以如果把重要的信息去除了,会降低最终返回结果的质量。

动态索引剪枝的方法主要发生在查询的处理阶段,主要是在查询的处理过程中,尽可能少地读取和处理那些对当前查询处理不重要的信息。为了能更好地估算出倒排链中那些对当前查询处理不太重要的信息,动态剪枝技术通常要求倒排链能按特定的顺序(如按 PageRank 分数)进行组织(Long X H et al. 2003),或者预先计算好一些信息(如每个词的最大分数等)。动态剪枝方法既可以采用剪枝过的索引来进行查询处理,也可以采用完成的索引来处理查询。一般情况下,动态索引剪枝方法返回的结果与穷尽查询处理是一样,不会降低搜索质量。图 8-13 是本文采用的索引剪枝方法的分类体系,接下来分别讲述静态索引剪枝和动态索引剪枝的方法。

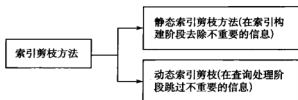


图 8-13 索引剪枝方法的分类

### 一、静态索引剪枝方法

静态索引剪枝是在倒排索引的构建过程中,去除那些在未来查询处理过程中很可能不会被用户关心或者说对用户不重要的信息,从而减少倒排链的长度。这种做法既节省存储空间,也减少查询处理的数据量。那些被去除的信息是在查询阶段无法还原,所以静态索引剪枝通常被认为是一种有损压缩算法。去除信息的方法的好坏直接影响着检索效果。静态索引剪枝方法的目的是要缩短图 8-3 中倒排链  $I_i$  的长度,从而减小倒排索引的最终大小。根据静态索引剪枝方法所关注的角度,主要有两类方法:倒排链为中心的剪枝方法(Term Centric Pruning, TCP)(Carmel et al. 2001)和文档为中心的剪枝方法(Document Centric Pruning, DCP)(Buttcher et al. 2006)。这两种方法分别从倒排链的角度和文档自身的角度出发,来去除那些不重要的信息。

#### 1. 倒排链为中心的剪枝(TCP)

TCP 剪枝方法(Carmel et al. 2001)关注的对象是每个索引词的倒排链  $I_i$ 。核心思想是首先对索引词  $t$  对应的倒排链  $I_i$  中的每个倒排项  $P_i$  按某种打分策略(如  $TF * IDF$ )进行打分,然后对  $I_i$  中的  $P_i$  按其分值进行排序得到第  $k$  大的  $P_k$  分值为  $s_t$ ,最后对分值小于  $s_t * \epsilon$  的  $P_i$  进行删除,其中  $\epsilon$  用于控制剪枝完成后的索引大小。算法 8-4 是对 TCP 法的一个描述。TCP 剪枝方法的主要问题是参数  $\epsilon$  的确定。 $\epsilon$  对于不同的索引词可以是一样的,也可以是不同的,这导致剪枝完成后的索引大小很难估算。另一种 TCP 剪枝方法对每个倒排链  $I_i$  保留固定数量的倒排项,如得分最高的 10K 个倒排项。其方法是对于长度小于 10K 的倒排链,不对其进行剪枝。对于长度大于 10K 的倒排链,保留分数最大的前 10K 个倒排项。通常这种保留固定数量的剪枝方法没有算法 1 采用的剪枝方法的效果好,但这种方法实现起来更加简单,而且剪枝后索引的大小也很容易控制。

#### 算法 8-4 TCP 静态索引剪枝方法

- 1: 输入:完整索引  $I$ ,  $k$ ,  $\epsilon$
- 2: 输出:剪枝完的索引  $I'$

```

3:  for each posting list  $I_i$  in  $I$  do
4:    if  $|I_i| > k$  then
5:      for each  $P_i$  in  $I_i$  do
6:        compute  $\text{Score}(t, P_i)$ 
7:        let  $s_i$  be the  $k$ th best entry in  $I_i$ ,  $\tau_i = s_i \cdot \epsilon$ 
8:      for each  $P_i$  in  $I_i$  do
9:        if  $\text{Score}(t, P_i) \leq \tau_i$  then
10:         remove  $P_i$  in  $I_i$ 
11:   save  $I_i$  in the pruned inverted index  $I'$ 

```

## 2. 文档为中心的剪枝(DCP)

DCP 剪枝方法关注的对象是每篇文档  $d$  本身。核心思想是首先对文档  $d$  中的词按某种打分策略(如 KL 散度)进行打分  $\text{Score}(t, d)$ , 然后为每篇文档保留得分最高的  $|d| * \lambda$  个词, 其中  $|d|$  是文档中包含不同词的个数,  $\lambda$  是保留词的比例, 算法 8-5 是 DCP 方法的算法伪码。在文献(Buttcher et al. 2006)的实验表明, DCP 法比 TCP 法的效果好, 而且只需要保留 10% 的倒排项时, 就能基本达到使用完整索引的检索效果。主要原因是在 TCP 方法中, 只能得到索引词自身在文档中的信息, 而 DCP 方法可以得到所有词在文档中的信息。因此 DCP 方法更容易保留那些文档中重要的信息。文献(Buttcher et al. 2006)还为每个文档保留固定数量的词, 但这种方法没有为每篇文档保留固定比例的方法的效果好。DCP 剪枝方法的一个问题是, 为了计算出文档中词的重要度分数, 需要在索引文档集中做一些预先的统计, 如统计  $F_i$  和  $f_i$ 。

### 算法 8-5 DCP 静态索引剪枝方法

```

1:  输入: 完整文档  $D$ ,  $\lambda$ 
2:  输出: 剪枝完的  $D'$ 
3:  for each document  $d$  in  $D$  do
4:    for each  $t$  in  $d$  do
5:      compute  $\text{Score}(t, d)$ 
6:      let  $k = \lambda \cdot |d|$ , let  $s_i$  be the  $k$  th best entry in  $d$ ,
7:      for each  $t$  in  $d$  do
8:        if  $\text{Score}(t, d) \leq s_i$  then
9:          remove  $t$  in  $d$ 
10:   save  $d$  in the pruned collection  $D'$ 

```

## 3. 其他静态索引剪枝方法

另外的一些静态索引剪枝方法主要是对 TCP 和 DCP 方法的扩展。文献(Nguyen 2009)直接对所有的倒排项  $P$  按某种打分策略做全局的重要性判断(在

TCP 法是在一个倒排链  $I_i$  范围内进行,在 DCP 法是在一个文档  $d$  范围内进行),然后取分值高的前  $k$  个倒排项放入剪枝后的倒排索引中。这种方法的主要问题是倒排项的全局分数比较难计算,另外实验表明(Nguyen 2009)这个方法的效果并不比 TCP 和 DCP 方法好。文献(Blanco et al. 2007, Skobeltsyn et al. 2008)采用删除整个倒排链的方法。首先对文档集中的一些停用词或信息量小的词进行识别,然后在索引剪枝时把这些词从索引中去除。这种剪枝方法相当于把一个词对应的倒排链整体删除,实验表明这种策略与 TCP 法效果相差不大(Blanco et al. 2007)。文献(Garcia 2007, Altingovde et al. 2009, Zheng et al. 2009)直接对整篇文档内容进行删除。在文献(Zheng et al. 2009)通过  $TF * IDF$  的方法为每篇文档计算一个分数,然后保存得分高的文档。文献(Zheng et al. 2009)则使用信息熵的方法对每个文档进行熵的计算,然后按大小对文档进行排序,在剪枝时保留熵值小的文档。而在文献(Garcia 2007, Altingovde et al. 2009)中,根据文档在检索阶段被访问的频率来对文档进行排序,剪枝的索引中保存那个频率高的文档。文献(Garcia 2007, Altingovde et al. 2009, Lam et al. 2010)利用已有的查询集的查询来预测出重要的词,然后保留那些经常在查询集中出现的词或者经常出现在 Top-K 查询结果中的文档。这种策略经常与 DCP 法或 TCP 法结合,能得到更好的剪枝效果。文献(Moura et al. 2008)在剪枝时,以文档中的一个句子为单位进行信息的保留。首先利用 TCP 法得到剪枝完的索引,然后对文档内的句子按 TCP 剪枝完保留下来的词的多少进行排序,最后只保留分值高的句子。这种剪枝策略能有效的保留查询的局部性(即查询词通常在同一句子里出现),所以这种剪枝策略能较好的支持短语查询和 AND 查询。文献(Blanco et al. 2010, Thota et al. 2011)利用了概率与假设检索的方法来更好地对词的重要度进行估算。文献(Blanco et al. 2007)讨论了在剪枝完的索引中,是否需要更新文档长度,平均文档长度等一些信息。文献(Ntoulas et al. 2007)讨论了如何结合剪枝后的索引与完整索引以得到查询结果与使用完整索引得到 Top-K 结果一致,并提升查询的处理速度。总之,静态索引剪枝方法的主要研究问题在于,如何在索引的构建阶段来预测信息的重要程度,以及应该去除多少信息。

## 二、动态索引剪枝方法

动态索引剪枝又称为查询提前截断,主要思想是在处理查询的过程中,尽量少的读取或处理查询词对应倒排链的数据。动态索引剪枝的优点是剪枝过程发生在查询处理阶段,这时知道的查询信息比较多,所以更加容易计算信息的重要度。另外动态索引剪枝的方法一般不会影响最终查询的效果。缺点是依赖于倒排索引的结构与排序函数,一般要求倒排索引中的倒排链按特定属性有序排列或者在倒排链中存储一些便于估算信息重要度的信息。在本小节中,首先介绍常用的查询处理方式和需要的索引结构,其次介绍在不同查询处理方式和排序函数下的剪枝方法。

### 1. 常用索引结构与查询处理方法

在信息检索领域中,提出了很多不同查询处理的方式,这些查询处理方式主要可以分成以下三类:

1) Document-At-A-Time(DAAT):在 DAAT 的查询处理过程中,查询词对应的倒排链都会被同时打开,然后同时遍历这些倒排链。每次对这些倒排链中文档号最小的文档计算查询词与文档的相关性分数。所有的查询词是被同时处理的,所以文档的分数在处理完一篇文档后就可以完整得到。因此,在查询处理的过程中,DAAT 方法只需要用一个优先队列来保存当前 Top-K 结果的文档号与分数。

2) Term-At-A-Time (TAAT):在 TAAT 的查询处理过程中,每次只打开一个查询词的倒排链并进行处理。每次只处理一个查询词,所以每处理完一篇文档后,只能得到这个词对这篇文档贡献的分数。只有当所有的查询词都处理完后,才能得到文档的完整分数。因此,TAAT 查询处理方法需要一个文档分数累加器数组来保存文档的临时分数。当文档集模型很大时,这个文档分数累加器的存储开销会变得非常大。

3) Score-At-A-Time (SAAT):与 TAAT 方法类似,SAAT 方法每次打开一个查询词倒排链中分数(如词频或影响值)相同的块,然后对这个块的文档进行处理。SAAT 方法也需要一个为每篇文档分配一个分数累加器来保存处理过程中产生的临时分数。与 TAAT 方法不同的是每次只处理一个词对应倒排链的一部分数据,这部分数据通常是当前倒排链中对文档的贡献值最大的部分。

TAAT 和 SAAT 在传统的信息检索中有很多研究,这些研究方法考虑倒排链的数据主要是从磁盘读取。在 TAAT 或 SAAT 的方法中,每次读取一个查询词的数据,就可以顺序的读取数据,所以能较大地提升读取数据的速度。在 DAAT 方法中,每个查询词的数据需要同时读取。在读取时,需要不停地在各个倒排链的数据中来回。虽然使用缓存技术可以减少数据来回从磁盘读取的次数,但还是不够高效。最近的研究主要集中在 DAAT 查询处理的方法。首先,现在计算机的内存容量已经很大,一般使用倒排链的缓存技术可以把大部分重要的数据放到内存中。当所有倒排链的数据在内存时,同时遍历倒排链数据就变成一件很容易的事。其次,在查询大规模的数据集时,DAAT 方法只需要一个空间需求很小优先队列结构来存放 Top-K 的查询结果。TAAT 或 SAAT 查询处理方法需要一个存储空间很大的分数累加器数组来存放文档的局部分数,其存储空间大小与文档集规模成正比。最后,DAAT 查询处理方法在处理需要查询词之间的关系的查询时(如短语查询),比 TAAT 或 SAAT 的方法容易。因为在 DAAT 方法中查询词在文档内的关联信息是一次就可以全部得到,而在 TAAT 或 DAAT 的查询处理过程中需要在分数累加器中保存查询词之间的关系,并且这些关系的维护代价是非常高的。所以,现代的搜索引擎基本上采用 DAAT 方式进行查询处理。

不同的查询处理方式需要的倒排索引结构不一样,根据倒排链中倒排项排列的

顺序不同,倒排索引结构主要可以分为以下两类:

1) Document-Sorted(Brin et al. 1998, Broder et al. 2003, Chakrabarti et al. 2011, Ding et al. 2011):在文档号排序索引中,倒排链中的倒排项的组织方式是按文档号  $d$  递增序进行组织。

2) Frequency or Impact-Sorted(Persin et al. 1996, Anh et al. 2001, Anh et al. 2006, Anh et al. 2006, Strohmaier et al. 2007)。在词频排序的索引中(Persin et al. 1996),倒排项首先按照词频大小把词频相同的倒排项分到同一组中,再对属于同一组的倒排项按文档号的递增序进行排列。在影响值的索引结构中(Anh et al. 2001, Anh et al. 2006),首先,在一篇文档范围内对每个词计算出这个词的重要程度,也就是影响值(impact)。影响值往往可以用一个值较小的正整数表示,如用 2 字节来表示(Brin et al. 1998, Altingovde et al. 2009),然后在倒排链中把包含相同影响值的倒排项分到同一块中,再对同一块的倒排项按文档号进行递增的排列。因此,按影响值排序的索引与按词频排序的索引的结构基本一致,不同点是影响值是已计算完的相关分数的近似,一般在检索阶段只需要对其相加即可,而词频索引只包含了词频信息,在检索阶段还是需要对其进行计算。

词频排序索引或影响值排序索引结构相对于文档号排序索引主要有以下局限性:①对于影响值索引,每个索引词对应文档的分数在索引阶段已被固定,不能在检索时调整。②词频或影响值排序的索引把倒排链分成多个不同权值块,不利于提升倒排索引的压缩率。③基于词频或影响值排序的索引的查询处理方式一般不能很好支持布尔查询、短语查询等其他类型的查询。④在词频排序索引或影响值排序索引中增加一篇新的文档比较困难。另外,词频或影响值排序的索引结构一般只适用于倒排项中只包含  $\langle d, f_{i,d} \rangle$  的索引结构。在现代的搜索引擎中主要使用按文档号排序的索引结构来处理查询。表 8-4 描述了各种索引组织结构及其可支持查询处理方法。文献(Long X H et al. 2003)提出 fancy-list 的索引结构组织方式,是词频索引或影响值索引的一个变种,主要思想是把倒排链分成两部分,在倒排链中的第一部分存放重要的文档,第二部分存放其他文档,每分部中文档仍按文档号递增进行组织。

表 8-4 不同索引的组织结构及其支持的查询处理方式

倒排项包含信息	倒排索引组织结构	可支持的查询处理方法
$\langle d \rangle$	Document-Sorted	DAAT, TAAT
$\langle d, f_{i,d} \rangle$	Document-Sorted Frequency or Impact-Sorted	DAAT, TAAT TAAT, SAAT
$\langle d, f_{i,d}, [pos_1, \dots, pos_{f_{i,d}}] \rangle$	Document-Sorted	DAAT, TAAT



## 2. 动态剪枝方法的分类

查询处理所需要的时间一般由读取相关查询词倒排链的数据时间,筛选候选的文档时间和对候选文档进行打分时间三部分组成。如果倒排链的数据已在内存,则读取相关查询词倒排链的数据这部分的时间可以忽略。因此,查询处理时间主要是筛选候选文档及对候选文档进行打分。根据动态剪枝的方法发生的条件,现有的剪枝方法大体上从以下四方面来提升查询的处理速度。

1) 提前结束查询处理:这种方法通常要求把重要的倒排项放在倒排链的前面,把不重要的倒排项放在倒排链的尾部,如按照词频或影响值排序的索引。查询处理的过程中,那些重要的文档能最先得到处理。当分数高的文档足够多或者查询处理停止条件满足时,就可以返回 Top-K 分数最高的文档。这里查询停止条件根据需要可以适当地改变,一般要求 Top-K 集合中,分数最高的第 K 个文档的分数大于其他不在 Top-K 集合中的文档的分数,并且在 Top-K 集合中文档之间的顺序与穷尽查询处理的方式一致。典型的方法是文献(Anh et al. 2006, Strohman et al. 2007)中基于影响值索引的剪枝方法。

2) 倒排链内数据的跳跃处理:当倒排链中的倒排项按文档号序列组织时,那些与查询最相关的文档有可能分布在倒排链中各个地方,因此提前结束查询处理的方法在这种情况下往往是无效的。如果能在倒排链中跳过那些权重比较低的倒排项,那也可以加快查询处理的速度,所以这类剪枝的方法通常要求倒排链中存储跳查表来帮助数据的跳跃处理。WAND(Broder et al. 2003)是这类算法中的一个典型,另外文献(Chakrabarti et al. 2011, Ding et al. 2011)基于块索引也是属于这一类剪枝方法。

3) 去除查询词:对于一些停用词,通过在查询处理之前会把这些词从查询中去除。另外,在 TAAT 的查询处理过程中,如果处理完多个查询词后发现是否处理其余的查询词不会影响最终的结果时,那么这些剩余的查询词就可以被完整的跳过。典型的方法就是 BL 方法(Buckley et al. 1985)和一些去停用词方法。

4) 尽早结束文档打分:文档的打分所花的时间查询处理的过程中也占据了很大一部分时间。这种剪枝方法在文档的打分过程中,通过已打分的词和未打分的词的最大分数对文档分数最大分数的估算来决定是否对文档进行完整的打分。典型的算法是基于 DAAT 查询处理的 MAXSCORE(Turtle et al. 1995, Campinas et al. 2011)算法。

这四类提升查询处理的方法是可以相互结合使用,以便更大地提升查询处理的速度。本文根据查询处理的方式,把剪枝方法分成三类:DAAT 剪枝方法、TAAT 剪枝方法和 SAAT 剪枝方法。接下来对这些剪枝方法进行介绍。然后根据排序函数所使用的信息不同,把剪枝方法分成三类:①只考虑单个查询词与文档相关度的分数的剪枝方法。②在①的基础上,考虑文档自身权重的剪枝方法。如考虑文档的 Page

Rank 值。③在②的基础上,考虑查询词之间的关系的剪枝方法,例如查询词之间的距离。

### 3. 按查询处理方式分类的剪枝方法

在这一小节中主要介绍在不同查询处理方式的剪枝方法。这些剪枝方法通常默认的排序函数如式(8-1)所示,一般只考虑查询词与文档的相关性,这个相关性分数可以通过  $TF * IDF$ (Anh et al. 2001),  $BM25$ (Robertson et al. 1998)或语言模型计算得到。

$$S(d, q) = \sum_{t \in q} \text{Score}(t, d) \quad (8-1)$$

#### (1) DAAT 剪枝方法

基于 DAAT 查询处理方式主要用于文档号排序索引。此外 DAAT 查询处理方式也是当前主流的查询处理方法。下面主要介绍在 DAAT 查询处理方式下,如何进行查询的动态剪枝来加快查询处理速度。MAXSCORE(Turtle et al. 1995, Anh et al. 2006, Jonassen et al. 2011)和 WAND(Broder et al. 2003, Ding et al. 2011)是这种剪枝方法下的两个最典型的算法,另外 Block-Max MAXSCORE(BMM)和 Block-Max WAND(BMW)是在最大块索引下对 MAXSCORE 和 WAND 方法的改进。Interval-Based 方法(Cummins et al. 2010)是另一种在最大块索引下快速 DAAT 查询处理的方法。

MAXSCORE 算法的剪枝主要是在倒排链数据的跳跃和尽早结束文档打分这两个方面来加快查询的处理。MAXSCORE 算法要求每个倒排链  $I_i$  都预先计算这个倒排链中的最大分数  $S(I_i)$ 。在处理查询过程中,首先对每个查询词对应的倒排链按其最大分数  $S(I_i)$  从大到小排序,这个排序在查询处理过程中是不变的。接着计算出一个最大分数累加数组  $A$ ,其内容是根据  $I_i$  在查询词倒排链中的位置来计算只包含这个词及其后面查询词的文档能得到最大的分数。使用这个数组可以对文档进行最大的分数估算,如果文档的当前分数加上未出现词的最小的分数都不超过当前阈值时,则对这篇文档的打分就可以停止,如算法 8-6 的第 8 行所示,所以 MAXSCORE 算法能尽早的结束对文档的打分。随后根据当前的阈值计算出必须包含查询词的集合  $T$ ,即如果一篇文档没有包含  $T$  中的词,则最大分数不会超过阈值,可以跳过对这篇文档的处理。接着从  $T$  选取当前最小的文档号  $d_{\text{cand}}$  作为候选文档进行打分。如果  $I_i$  对应的查询词不在  $T$  中,则  $I_i$  可以直接跳到文档号  $d_{\text{cand}}$  进行处理(第 11 行),这时可以跳过一部分  $I_i$  对应倒排链的数据。图 8-14 是 MAXSCORE 算法选择文档的一个示例,这里总共有 3 个倒排链  $\{T_1, T_2, T_3\}$ ,最大分数分别是 5, 3, 2,假定当前的阈值分数是 4,则



图 8-14 MAXSCORE 算法的示例

这时的  $T = \{T_1, T_2\}$ , 所以当前的选文档号应该是 11, 而  $T_3$  则直接可以跳过文档 5, 7。

算法 8-6 DAAT 查询处理下 MAXSCORE 算法

```

1: 输入: 按  $I_i$  最大的分数  $S(I_i)$  递减排序倒排链遍历器  $I = \{I_1, I_2 \dots I_n\}, n = |q|, k$ 
2: 输出: 前  $k$  分值最大的文档号及其分数
3:  $r = |I|$ ;  $resHeap = \{\}$ ; // 变量  $r$  是最小包含查询词的个数
4: 计算查询词最大累加分数数组  $A, A[I_i] = \sum_{i \leq j \leq |I|} S(I_j)$ 
5: while  $|I| > 0$  and  $r > 0$  do
6:    $score \leftarrow 0$ ;  $d_{and} \leftarrow \min_{i \leq r} (I_i.doc)$ ;
7:   for  $i \leftarrow 1$ ;  $i \leq |I|$ ;  $i \leftarrow i+1$  do
8:     if  $score + A[I_i] < resHeap.minScore$  then
9:       break
10:    if  $i > r$  then
11:      if  $skipTo(I_i, d_{and}) = \text{false}$  then // skipTo 指跳到大于等于文档号  $d_{and}$ 
12:        从  $I$  中删除  $I_i$ , 同时更新  $A, r$ ,
13:        continue
14:    if  $I_i.doc = d_{and}$  then
15:       $score \leftarrow score + score(I_i)$ 
16:      if  $score > resHeap.minScore$  then
17:         $resHeap.insert(d_{and}, score)$ ;
18:      for  $i \leftarrow r$ ;  $i > 1$ ;  $i \leftarrow i-1$  do
19:        if  $A[I_i] < resHeap.minScore$  then
20:           $r \leftarrow r-1$ 
21:      for  $i \leftarrow 1$ ;  $i \leq |I|$ ;  $i \leftarrow i+1$  do
22:        if  $I_i.doc = d_{and}$  and  $next(I_i) = \text{false}$  then
23:          从  $I$  中删除  $I_i$ , 同时更新  $A, r$ ;  $I \leftarrow i-1$ 
24: return  $resHeap.result()$ ;

```

WAND 算法的剪枝主要通过跳过倒排链的数据来提升查询的处理速度。与 MAXSCORE 算法一样, WAND 算法也要求每个倒排链  $I_i$  都预先计算这个倒排链中的最大分数  $S(I_i)$ 。WAND 处理查询的过程如下: ①对索引词对应的倒排链, 根据当前的文档号  $d_i$  按从小到大序列进行排序。②计算支点词与支点文档号。支点词计算过程如下: 对当前按文档号排序好的倒排链对应最大分数  $S(I_i)$  进行累加, 当累加的分数首次超过阈值时, 最后加入的词就是支点词。支点词对应倒排链的当前文档号就是支点文档号。③判断第一个词对应的文档号与支点文档号是否相同, 如果相同, 则对支点文档号对应的文档进行打分。如果不相同, 对于位置支点词前的倒排链直接跳到支点文档号, 然后再重复步骤①, ②直到处理完所有倒排链的数据。注意在 WAND 中每次都需要对倒排链根据当前的文档号进行排序。而在 MAX-

SCORE 算法中,只需要对倒排链的权重一次排序即可。在 WAND 进行提前结束文档打分相对 MAXSCORE 代价高。主要原因是在最大分数累加数组 A 中的值每处理一篇文档后就会变化,而 MAXSCORE 中是不变的。算法 8-7 是对 WAND 算法的描述,跳过倒排链数据主要集中在第 23,24 行。图 8-15 是 WAND 算法如果选择打分文档的一个示例。图中包含四个查询词的倒排链 $\{t_1, t_2, t_3, t_4\}$ 最大分数分别是 2, 1, 4, 3。在候选文档的选择过程中,首先对倒排链按文档号递增排序得到当前的文档号序列是 $\langle 11, 11, 22, 23 \rangle$ 。其次,计算支点词及文档,这里  $t_3$  是支点词支点文档号 22,  $t_1 + t_2 + t_3 = 7$ ,首次大于阈值 6,但这里第一个倒排链对应的文档号 11 与支点文档号不相等,所以其对  $t_1, t_2$  两个倒排链跳到 22 号文档,然后再重新计算支点词与支点文档号。

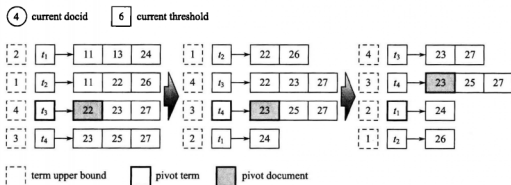


图 8-15 WAND 算法选择候选文档的过程

#### 算法 8-7 WAND 算法

- 1: 输入: 倒排链遍历器  $I = \{I_1, I_2, \dots, I_n\}, n = |q|, k$
- 2: 输出: 前  $k$  分值最大的文档号及其分数
- 3:  $resHeap \leftarrow \{\}$ ;
- 4: **while**  $|I| > 0$  **do**
- 5:      $sort(I)$  //sort the list by its current docIDs
- 6:      $pivot \leftarrow 1; s \leftarrow 0;$
- 7:     **for**  $pivot \leq |I|; pivot \leftarrow pivot + 1$  **do**
- 8:         **if**  $s + I_{pivot}.maxScore \geq resHeap.minScore$  **then**
- 9:             **break**;
- 10:         **else**  $s \leftarrow s + I_{pivot}.maxScore$
- 11:     **if**  $pivot > |I|$  **then**
- 12:         **return**  $resHeap.result()$ ;
- 13:     **if**  $I_1.curID = I_{pivot}.curID$  **then**
- 14:          $docScore \leftarrow 0;$
- 15:     **for**  $i \leftarrow 1; i \leq |I|; i \leftarrow i + 1$  **do**

```

16:  if  $I_i.curID = I_{pivot}.curID$  then
17:     $docScore \leftarrow docScore + score(I_i)$ 
18:     $next(I_i)$ ;
19:    if  $docScore > resHeap.minScore$  then
20:       $resHeap.insert(I_{pivot}.curID, docScore)$ ;
21:    else
22:      for  $i \leftarrow 1; i < pivot; i \leftarrow i+1$  do
23:        skip to( $I_i, I_{pivot}.curID$ );
24:      return  $resHeap.result()$ ;

```

MAXSCORE 和 WAND 剪枝算法在 DAAT 的查询处理方式下能一定程度地提升查询的处理速度。文献(Ding et al. 2011)提出了最大块索引(Block-Max Index)来进一步提升 WAND 查询的效率,这个算法称为 Block-Max WAND 或 BMW。在最大块索引中,倒排链仍是按照文档号递增的顺序进行存储,倒排项是按块进行组织,一个块可以是变长的也可以是定长的,如每个块由 128 个倒排项组成。这种组织方式的好处:①每个块都可以单独的压缩与解压缩;②同类型的数据能被一起压缩以达到更大的压缩率,同时对词频与位置信息的数据可以在需要时被解压;③对于每个块,可以保存一些块信息,可以快速地估计块中文档的相关性。在图 8-4 中,跳查表由跳查指针(skip pointer)组成。跳查指针在最大块索引中是一个三元组 $\langle d, p, BlockMaxScore \rangle$ ,表示在倒排链的  $p$  位置有一个以文档  $d$  开始的块,块中的最大的 IR 分数是 BlockMaxScore。

基于最大块索引结构,WAND 算法在找到支点文档后,可以利用每个倒排链当前块中的 BlockMaxScore 值来对文档的分数进行估算。这个值往往比倒排链中的最大分数要小很多,因此估算出来的文档最大分数相对会比较精确。如果估算出来的文档分数大于当前的阈值,则对文档进行正常的处理。如果估算出来文档的分数小于等于当前的阈值,则可以选择一个更大的候选文档进行打分。图 8-16 是基于最大块索引的支点文档号的选择示例。假定当前的支点文档号是 266,如果使用当前块的最大分数  $d_1 + d_2 + d_3 + d_4$  小于阈值,则在 WAND 算法中会选择 266 作为下一个候选文档,而最大块索引中,可以选择  $d_2 + 1$  作为下一个候选文档。这样每个块可以跳过更大的数据。

基于最大块索引的结构同样也适用于 MAXSCORE 算法,只需要在算法 8-6 中的第 7 行前加入利用块的最大信息估算文档当前的最大分数即可,如果估算的分数大于阈值,那么就进行正常的处理,否则按照 Block-Max WAND 的方法选一个更好的候选文档。称这种算法为 Block-Max MaxScore 或 BMM。

文献(Cummins et al. 2010)在最大块索引上,提出了 Interval-Base 的剪枝方法来高效的处理查询。首先,根据最大块信息 SI(SI 是一个三元组 $\langle \min DocID, \max DocID, Score \rangle$ ,表示块中的最小文档号、最大文档号以及块的最大分数)最大

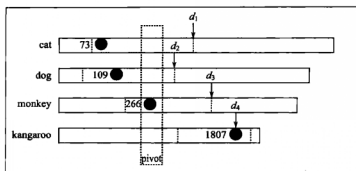


图 8-16 基于最大块索引的支点文档号的选择示例

最小文档号把整个文档区间划分成多个子区间,每个子区间最多跨一个块。然后,利用块的最大分数 Score 来估算每个子区间中文档的最大分数。最后,对每个区间按照正常的 DAAT 查询的处理方法进行处理。图 8-17 展示了在 Interval-Base 方法中文档子区间划分及子区间最大分数估算的一个示例。在图中共有三个词  $q_1$ ,  $q_2$ ,  $q_3$ , 划分完的子区间使用黑体数字表示。当划分完子区间后,可以先对包含最重文档的子区间进行处理。如果剩余子区间的最大分数小于当前阈值时,整个查询就可以停止。所以 Interval-Base 方法主要是利用查询提前结束方式来加快查询的处理。

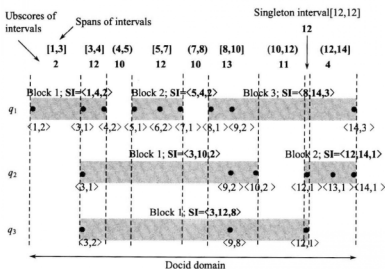


图 8-17 Interval-Base 剪枝方法中文档子区间划分的示例

## (2) TAAT 剪枝方法

文献(Buckley et al. 1985)首先在 TAAT<sup>†</sup> 的查询处理的方法中引用了剪枝的

方法,主要是对一些分数贡献低的词的倒排链进行删除,从而节省查询时间。TAAT 剪枝方法的思路主要集中在如何使用更少的分数累加器(即减少查询处理过程中内存的消耗)来加快查询的处理速度。使用 TAAT 方式查询时,首先对倒排链按从短到长进行排序,然后依次对每个倒排链进行处理。在处理的过程中,需要为每个在倒排链中出现的文档使用一个分数累加器  $A_d$  来保存文档的部分分数。当处理完所有的倒排链后,所有累加器中的分数会根据文档的长度进行正规化处理,然后返回  $k$  分数最高的文档号。这种查询的处理方式如果查询时包含常用词,需要累加器的数量非常大。一般的做法是每篇文档预先分配好一个累加器数组,第  $i$  号文档对应累加器数组中第  $i$  个元素。如果一个文档集包含 25MB 文档,那么在查询处理的过程中,需要近 100MB 的存储空间来保存临时分数,同时处理这么大的数组效率也非常低。文献(Turtle et al. 1995)使用 MAXSCORE 的策略来决定是否对分数累加器进行保留,如果文档当前分数加上未打分词的最大分数都不能超过 Top-K 阈值分数时,则把这个文档对应的分数累加器去除,否则继续保留累加器。文献(Moffat et al. 1996)提出了 quit 和 continue 两种策略来对累加器的数量进行剪枝。做法如下:

1) 在 quit 策略中,首先确定目标使用的累加器数量  $L$ ,并初始化累加器集合  $A$ 。其次对每个倒排链  $I_i$  进行依次处理,对于  $d \in I_i$ ,如果  $A_d$  已在  $A$  中,则更新  $A_d$  的分数,如果不在,则把  $A_d$  加入  $A$  中。有两种方法判断当前已有累加器的数量  $|A|$  是否超过  $L$  的限制。quit-full 方法是每处理完一个  $I_i$  进行判断,quit-part 方法是每处理完一篇文档进行判断。

2) 在 continue 策略中,如果累加器的数量已达到目标使用的累加器数量  $L$ ,则不再增加累加器,只对那些已在  $A$  中的文档进行分数的更新。

实验表明(Moffat et al. 1996),这种对累加器数量剪枝的方法能加快查询的处理速度,并且对返回 Top-K 的结果质量的影响非常小。但是 quit 和 continue 的处理方式在当前搜索引擎的查询中不是很高效,原因是这种方法适用于查询词多的查询,而搜索引擎中的查询都非常短。文献(Lester et al. 2005)提出了 adaptive-pruning 的方法来更好的控制查询时累加器使用的数量。做法是在处理倒排链  $I_i$  之前,会根据一些信息,如当前累加器的数量  $|A|$ ,最大可使用累加器的数量  $|L|$  和  $I_i$  的长度等,估算一个阈值  $T$ ,只有那些分数大于  $T$  的文档才能新增累加器。分数小于  $T$  的文档只能对已有累加器的分数进行更新。同时,在文档的处理过程中,会删除那些分数较低的累加器。文献(Jonassen et al. 2011)在文献(Lester et al. 2005)的基础上加入跳查信息,使得在处理的过程中,可以跳过那些不重要的数据块。具体算法如算法 8-8 所示。在第 6 行中,它会判断最小的词频  $h$  是否大小当前处理倒排链中最大的词频值  $f_{\max}$ 。如果是,那么对这个倒排链不需要加入新的累加器,只需要对那些已有累加器的文档进行分数累加,在倒排链中直接跳到相应的文档即可,否则就按正常的流程进行处理。

算法 8-8 支持跳查的 Adaptive Pruning 算法

```

1: 输入: 按  $F_i$  递减排序的倒排链遍历器  $I = \{I_1, I_2, \dots, I_n\}$ , 目标累加器数量  $L$ , 返加的文
   档数  $k$ 
2: 输出: 前  $k$  分值最大的文档号及其分数
3:  $A \leftarrow \{\}$ ;  $T \leftarrow 0.0$ ;  $h \leftarrow 0$ 
4: for  $i \leftarrow 1$ ;  $i \leq |I|$ ;  $i \leftarrow i+1$  do
5: 利用  $L, |A|, F_i$  和先前的  $T$  计算新的  $T$  和  $h$  值// $h$  是对词频的最小值
6: if  $h > f_{\max}$  then
7:   foreach  $A_d \in A$  do
8:     if  $I_i.docID < d$  then
9:       if  $skipTo(I_i) = false$  then  $A \leftarrow A - \{A' | A' < T\}$ ; break;
10:    if  $I_i.docID = d$  then  $A_d \leftarrow A_d + Score(I_i)$ 
11:    if  $A_d < T$  then  $A \leftarrow A - A_d$ 
12:  else
13:    foreach  $d \in I_i \cup A$  do
14:      if  $d \notin I_i$  then
15:        if  $A_d < T$  then  $A \leftarrow A - A_d$ 
16:      elseif  $d \notin A$  then
17:        if  $I_i.freq \geq h$  then
18:           $d \leftarrow I_i.docID$ ;  $A_d \leftarrow score(I_i)$ ;  $A \leftarrow A + A_d$ 
19:        else
20:           $A_d \leftarrow A_d + score(I_i)$ ; if  $A_d < T$  then  $A \leftarrow A - A_d$ 
21:  return  $A$  中分数最高的  $k$  个文档

```

### (3) SAAT 剪枝方法

SAAT 查询处理方式只能在影响值索引上进行, 文献(Anh et al. 2006)首次提出 SAAT 查询处理的剪枝方法。在这种剪枝方法中, 总共定义四类查询处理的模式。

1) OR: 在 OR 模式下, 所有的文档都会被打分, 并为新出现的文档分配分数累加器。

2) AND: 在 AND 模式下, 不再为新出现的文档分配新的分数累加器。只对那些有累加器的文档进行分数的累加。

3) REFINER: 当 Top-K 的文档已被确定, 但这些文档在最终的 Top-K 结果中的顺序还没有确定时使用 REFINER 模式。这时, 只对 Top-K 内的文档进行分数的累加。

4) IGNORE: 在 IGNORE 模式下, 所有的文档都不会被考虑, 停止处理查询。

在影响值索引中, 每个倒排链最多包含  $k$  个不同影响值的块。这里用  $w_{d,i}$  表示一个块中文档的影响值, 范围是  $[1, k]$ , 每个块的处理模式只能是上述四种模式的一种。查询词在查询中对应的影响值的范围也被固定在  $[1, k]$  之间, 用  $w_{q,i}$  表示。查



询处理过程如下。

步骤 1: 在查询开始阶段, 首先对各个查询词对应倒排链中分数高的块用 OR 模型进行处理。

步骤 2: 当  $S \geq \sum \{next_t \mid t \in q\}$  满足时, 即  $S$  比未出现过的文档的最大分数大时, 进行 AND 模式处理查询。这里  $next_t$  表示  $t$  对应倒排链中未处理块的最大分数,  $S$  是当前 Top-K 文档集合  $R$  中最小的分数。

步骤 3: 当  $S \geq \max\{M_d \mid d \in A, d \notin R\}$  满足时, 进行 REFINER 模型处理查询。这里  $M_d$  是文档  $d$  的最大分数, 计算方式如下:  $M_d = A_d + \sum \{next_t \mid t \in q, t \notin T_d\}$ , 其中  $A$  是当前已有的累加器集合, 这里的累加器不仅有文档号和分数, 还在已计算过词的集合  $T_d$ 。

步骤 4: 当  $A_i \geq M_i$ , ( $i \in R$ ) 时, 查询处理进行 IGNORE 模式, 此时查询就可以提前停止。

算法 8-9 是对 SAAT 剪枝方法的描述, 首先对查询词对应的所有块按影响值大小进行排序, 然后按上述步骤进行查询处理, 每次从未处理的块中获得影响值最大的块进行处理。图 8-18 是 SAAT 剪枝方法在基于影响值索引上进行查询处理时, 所经历的查询处理模式及使用分数累加器数量的变化情况。从图中可以看出, 在 OR 模式下, 累加器数量是不断增加的。当进入 AND 模式后, 累加器数量开始处于稳定状态, 不再增加新的累加器。当进入 REFINER 模式后, 累加器的数量会减少到  $|R|$ 。文献 (Strohman et al. 2007) 对 SAAT 剪枝算法中的 AND 查询处理模型进行修改, 主要是对这一过程中的累加器数量进行剪枝。另外, 块中还加入了跳查表, 可以加快在 AND 模式下查询处理速度。因此, SAAT 的剪枝方法主要利用了查询提前停止, 倒排链内数据的跳跃及文档提前结束打分来提升查询的处理速度。文献 (Anh et al. 2006) 讨论了通过 SAAT 剪枝方法来提升布尔查询的速度。

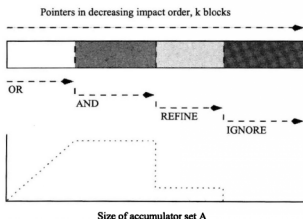


图 8-18 SAAT 方法处理查询处理模式及分数累加器数量的变化

算法 8-9 SAAT 剪枝方法

```

1: 输入: 按 Impact 值排序的倒排链遍历器  $I = \{I_1, I_2, \dots, I_n\}$ , 返回的文档数  $k$ 
2: 输出: 前  $k$  分值最大的文档号及其分数
3:  $A \leftarrow \{\}; T \leftarrow \{\}; R \leftarrow \{\}; mode \leftarrow OR$ 
4: 对每个词  $t$  计算  $next_t$ 
5: for  $i \leftarrow k^2; i > 0; i \leftarrow i-1$  do
6:   foreach  $\langle t, d, w_{d,t} \rangle$  in block  $i = w_{d,t} * w_{q,t}$ , do
7:     if  $mode = OR$  or  $A_d > 0$  then
8:        $A_d \leftarrow A_d + i; T_d \leftarrow T_d \cup \{t\}$ 
9:   更新  $next_t$  值
10:  if  $S \geq \sum \{next_t | t \in q\}$  then
11:     $mode \leftarrow AND$ 
12:  if  $S \geq \max\{M_d | d \in A, d \notin R\}$  then
13:     $mode \leftarrow REFIN$ 
14:    foreach  $d \in A$  and  $d \notin R$  do
15:       $A \leftarrow A - A_d$ 
16:  if  $A_j \geq M_j, (j \in R)$  then
17:    break;
18: return  $A$  中分数最高的  $k$  个文档

```

#### 4. 按照排序函数分类的剪枝方法介绍

将对排序函数包含更多的特征的剪枝方法进行讨论。前面介绍了基于各种查询处理方式的剪枝方法,但都假定文档的排序函数只包含查询词与文档的相关性分数,在实际的系统中,排序函数会比较复杂。一个比较常用的排序函数为

$$S(d, q) = \alpha \cdot SR(d) + \beta \cdot IR(d, q) + \gamma \cdot TP(d, q) \quad (8-2)$$

其中,  $S(d, q)$  是文档  $d$  对于查询  $q$  的整个分数,  $SR(d)$  是文档的静态分数,如 Page-Rank 值,文档的质量等。  $IR(d, q)$  是查询与文档的相关性分数,其可由式(8-1)计算得到。  $TP(d, q)$  是查询词在文档  $d$  在相互关系的分数,如查询词接近程度的分数。  $\alpha, \beta, \gamma$  分别是各部分分数对整个分数的贡献的比例因子,且满足  $\alpha + \beta + \gamma = 1$ 。另外,  $SR, IR$  和  $TP$  的分数被归一化处理。下面主要讨论使用此公式作为排序函数时,如何高效地进行索引的动态剪枝。

##### (1) $SR + IR$ 剪枝方法

在这小节中,主要讨论当排序函数包含  $SR$  和  $IR$  分数时,如何快速的对查询进行剪枝。在 DAAT, TAAT 和 SAAT 查询处理过程中,通常使用文档当前的部分分数加上未打分过词的最大分数来估算文档的最大分数。当排序函数加入  $SR$  后,对文档的最大分数估算时还需要加入  $SR$  这个分数。因此,如何组织倒排索引结构来更好的估算文档的最大分数是这类剪枝方法的关键问题。

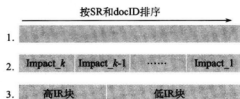


图 8-19 当前支持高效 SR+IR  
剪枝的索引结构

文献(Long et al. 2003)提出了按 SR 分数对文档号进行重排序的方法。首先对所有的文档按照 SR 进行全局排序,其次对原始索引中的文档号用全局文档号进行代替,然后对倒排链按照文档号进行排序。如图 8-19 所示,当前主要有 3 种对倒排链的组织:①整个倒排链接 SR 和文档号排序;②影响值索引中各个块内按

SR 和文档号排序;③fancy list 索引结构,第一部分包含 IR 分值高的文档,第二部分包含 IR 分值低的文档,这两部分中的文档按 SR 和 docID 进行排序。另外,文献(Zhang et al. 2010)提出了一种按文档中最大词的 IR 分数与文档的 SR 分数结合的方法,然后按这个结合后的分数进行文档的重排序,其索引结构与①类似。

对于第一种索引结构的剪枝算法(Long et al. 2003, Zhu et al. 2007, Zhang et al. 2010),基本考虑点是因为排序函数中包含 SR 分数,那些 SR 大的文档就很可能出现在最后的 Top-K 的结果中,因此把 SR 值大的文档放在倒排链的前面部分有利于查询的提前结束。DAAT 剪枝方法都适用于这种索引组织结构,只需要在估算文档最大的分数时,加上当前文档的 SR 值即可。算法 8-10 是基于 SR 排序的常用剪枝方法。第 6 行可以使用 DAAT 的剪枝方法计算一个文档的分数。第 10 行中的  $S_i$  是当前未处理的文档能达到的最大分数,可以用式 8-3 计算得到,当  $S_k \geq S_i$  成立时,即 Top-K 中的文档的分数都比未处理文档的最大分数还大时,查询就可以提前结束。

$$S_i = \alpha \cdot SR_{cur} + \beta \cdot IR_{max} \quad (8-3)$$

算法 8-10 按 SR 排序索引的剪枝方法

```

1: 输入:按 SR 值排序的倒排链遍历器  $I = \{I_1, I_2, \dots, I_n\}$ , 返回的文档数  $k$ 
2: 输出:前  $k$  分值最大的文档号及其分数
3:  $R \leftarrow \{\}$  //当前 Top-K 文档集合
4:  $S_k \leftarrow \{\}$  //在 R 集合中最小的分数
5: for each doc  $d$  in  $I$  do
6:   evaluate  $d$ 
7:   if  $|R| < k$  or  $d.score > S_k$  then
8:      $R.insert(d)$ 
9:      $S_k \leftarrow R_{min}$ 
10:  update  $S_i // S_i = \alpha \cdot SR_{cur} + \beta \cdot IR_{max}$ 
11: if  $|R| \geq k$  and  $S_k \geq S_i$  then
12:   return  $R$ ;
13: return  $R$ 

```

对于第②、③两种索引结构,可以使用 SAAT 的剪枝方法进行快速的查询处理,算法 8-11 是基于这种索引结构的剪枝方法,每次处理其中的一个块。每处理一个文档,会估算这个文档的最大分数与最小分数(主要是根据已打过分的词与未打过分的词进行计算)。然后用文档最大分数,最小分数是否大于阈值  $S_i$  来判断是否为此文档分配累加器。当文档的最大分数小于阈值  $S_i$  时,则把这个文档对应的累加器删除。其中,  $S_i = \max\{S_1, S_2\}$ ,  $S_1$  和  $S_2$  可以根据式(8-4)和式(8-5)进行估算。

$$S_1 = \alpha \cdot SR_{cur} + \beta \cdot \sum_{t \in q} T_{\max(t, \text{seg})} \quad (8-4)$$

$$S_2 = \alpha \cdot SR_{\max} + \beta \cdot \sum_{t \in q} T_{\max(t, \text{seg}+1)} \quad (8-5)$$

算法 8-11 多索引块的剪枝方法

```

1: 输入:按 SR 值排序的倒排链遍历器  $I = \{I_1, I_2, \dots, I_n\}$ , 返回的文档数  $k$ 
2: 输出:前  $k$  分值最大的文档号及其分数
3:  $R \leftarrow \{\}$  //当前 Top-K 文档集合
4:  $S_k \leftarrow \{\}$  //在 R 集中最小的分数
5:  $A \leftarrow \{\}$  //候选文档的分数累加器
6: for each segment seg do
7:   for each doc  $d$  in seg do
8:     evaluate  $d$  // 计算  $d$ .minScore 和  $d$ .maxScore
9:     if  $|R| < k$  or  $d$ .minScore  $> S_k$  then
10:        $R$ .insert( $d$ )
11:       if !  $A$ .contain( $d$ ) and  $d$ .maxScore  $> S_k$  then
12:          $A$ .insert( $d$ )
13:       if  $A$ .contain( $d$ ) and  $d$ .maxScore  $\leq S_k$  then
14:          $A$ .remove( $d$ )
15:     compute  $S_i$ 
16:     if  $|R| \geq k$  and  $S_k \geq S_i$  and  $A-R = \emptyset$  then
17:       return  $R$ ;
18: return  $R$ 

```

## (2) SR + IR + TP 剪枝方法

考虑 TP 分数的剪枝方法相对比较少(Brin et al. 1998, Schenkel et al. 2007, Zhu et al. 2007, Zhu et al. 2008, Yan et al. 2010)。当排序函数加入 TP 分数,最大的问题是精确估算文档的最大分数变得更加困难,而且单独计算这个问题的代价也很大,往往需要访问倒排链中的位置序列,大小是文档序列的 3~5 倍。现有的方法主要可以分为以下两类。

1) 附索引(Schenkel et al. 2007; Zhu et al. 2008; Yan et al. 2010):这些方法主要是在倒排索引中增加辅助索引结构如短语索引、词组合索引等。在查询

处理中,首先利用辅助索引快速计算 TP 的分数高的文档列表,然后再利用图 8-19 中的倒排索引结构来进行查询的剪枝。这些方法的缺点是辅助索引结构占用的存储空间往往比较大,因此如何有效地减小辅助结构的大小是这些方法需要解决的问题。

2) 网页文本的结构信息(Brin et al. 1998, Zhu et al. 2007)。这些方法认为网页的一部分内容,如标题、锚文本中比其他部分的内容(如正文)在计算相关性分数时权重最高,而且这些权重要的文本往往也比较短。所以在查询处理时,首先在标题等倒排链短的内容上作完整的检索,然后在正文等倒排链长的文本上做剪枝。剪枝的处理方式与算法 8 类似。

### (3) 其他剪枝方法

在数据库领域中,也提出一些高效计算 Top-K 记录的算法。例如,在文献(Fagin 2003)中,提出 TA 与 NRA 两种算法,前者假定数据可以随机访问,后者假定数据只能顺序访问。文献(Ilyas 2008)对数据库领域的方法进行了综述,但这些方法主要的排序函数是式(8-4)。

由于加入 SR 或 TP 索引剪枝方法变的比较复杂,文献(Broder et al. 2003, Dean et al. 2004, Yan et al. 2009)提出了多阶段的文档排序方式,如首先使用 IR+SR 的剪枝方法快速的找出前  $K'$  个文档,然后利用更复杂的排序函数对  $K'$  个文档进行重排序返回  $K$  个文档。这些方法通常是非精确剪枝方法,即返回的结果与使用完整检索方法的返回结果是不一致的,但两者的差异非常小,基本不影响最终返回结果的质量。

## 第五节 混合索引技术

大量的统计研究表明,搜索引擎用户输入查询长度平均较短,并且很少使用系统提供的查询操作符。这种情况下,检索结果排序考虑用户输入的查询词之间的短语关系或者位置邻近关系,对提高检索结果的效果十分重要。通过丰富倒排文件的数据结构内容,这样的关系有可能在预处理阶段充分地表达出来,从而为检索服务算法的运行提供数据基础。文献(Anh et al. 2002)中介绍倒排索引的几种常见级别和索引的压缩技术,其中词级(Word-Level)的倒排索引记录索引词在文档中出现的每个位置信息,检索时通过这些位置信息来执行短语或邻近关系的检查。词级索引是倒排文件实现短语或邻近查询的一般组织方式。文献(Sadakane et al. 2001)提出了邻近查询的通用算法,给出了最短邻近距离定义下的最优解算法。不过这个算法开销太大,在实际系统中并不实用。文献(Brin et al. 1998)介绍了邻近查询的一种近似实现方法,具有更好的检索效率。文献(Bahle et al. 2002, Williams et al. 1999)提出了一种新的短语索引技术,对倒排索引词典里每个索引词,按其后续词组织倒排数据项,即为每个索引词与其后继的索引词建立辅助倒排索引。这种方法可以提高短语查询的效率,缺点是基本索引词典和后继词词典分开存储,在查询过程中

需要增加一次对后续词的词典数据读取,一定程度上抵消了对短语查询效率的提高。在中文信息检索领域,什么是最佳的索引单位一直是困扰中文全文检索的一个问题。研究表明使用中文分词,按词索引结合二元组(Bi-gram)索引是检索效率和效果较优的索引方式。这实际上是一种混合索引,不过对大规模文档集合,二元组索引的倒排索引词典膨胀迅速,索引文件中包含大量无用内容,对检索效率也带来有负面影响。在天网搜索引擎的实践中,针对上述各种技术的优劣,采用了一种基于未登录词自动识别技术的混合索引方法。实践表明,这一方法可以有效提高搜索引擎检索效率。

### 一、混合索引的原理

混合索引是在建立倒排索引过程中的一种索引词选择方法与技术。索引词的选择是检索系统实现的一个重要环节。现代搜索引擎普遍采用全文索引技术,把网页文档中提取出来的所有词语都选择参与索引。在理想情况下,索引词应该是表达文档内容的语义单位,对应着语言学里的词汇词的概念,它是专门表示含义,而其实际意义无法由组合成分相加得到的最小语言单位。但对于自动文档索引过程,识别文档中的词汇词,如短语十分困难,因此通常选取语法意义上的最小语言单位为索引词。对英文文档,这一过程相对容易。对中文网页文档的索引过程,词间没有空格自然分隔,为索引词的选择带来了新问题。文献(Nie et al. 2000)研究表明相对于按字索引,基于自动分词的索引方式具有良好的检索效果。对搜索引擎系统,更出于检索效率的因素,通常通过自动分词来选择索引词。在文档索引过程中,先通过中文自动分词程序的处理,把文档正文分割成为独立的分词单位,然后在这些分词单位基础上选择索引词。分词单位是指具有确定语义或语法功能的基本单位,通常被直接选作索引词。

目前,中文自动分词的成熟技术都是基于分词词典的机械型分词方法(刘开瑛 2000),这一方法的主要难点在于分词歧义处理和未登录词的识别,其分词词典规模是制约分词精度的重要因素。我们使用北京大学计算语言所的中文自动分词软件,该分词软件的基本词典规模为 6 万词条。中文自动分词软件使用的词典选词十分严格,随意加入新词将影响分词软件的歧义处理过程,导致分词精度下降。文献(Stokoe et al. 2003)研究表明对英文文档数据,信息检索系统加入词歧义处理可以提高检索精度。对于中文文本,这一问题更加突出,任意扩大分词词典规模而忽略对分词精度的影响会对检索系统的检索效果带来负作用。同时,对处理 Web 数据,分词基本词典的规模是远远不够的。一方面,网上大量的常用词、新出现词、专业词汇等没有被收录,从而会被分词程序切分成分离的单字,每个单字被分别索引。这样的词在检索时会按短语查询执行,虽然可以检索出基本相同的结果集体,但执行过程需要从倒排文件中读取多个索引词的倒排项数据,然后执行位置检查,这大大降低了系统的检索效率。另一方面,分词词典中的分词单位一般很短,常用的短语也会被分词程序切分开,同样这一方式在对短语的查询上效率很低。如果分词程序使用的词

典中分词单位过长,切分出短语,又可能使得组成短语的词无法被检索,导致检索系统召回率下降。如何扩大分词词典的规模,使得分词程序能够切分出更多的词,甚至短语,同时又不降低分词程序的分词精度,不降低检索效果是中文搜索引擎检索系统面临的一个基本问题。

天网检索系统采用混合索引技术解决上述问题(彭波 2004a)。这一技术首先用统计方法对索引文档中的未登录词进行识别,把识别出的新词(不被基本词典收录的字串)放入一个扩展词典。这可以有效扩大词典规模,但由于统计方法识别未登录词存在相当的错误率,扩展词典里面也存在不少被错误识别的词。系统目前控制扩展词典规模在 50 万词条左右。扩展词典在保存时,把识别的新词词条使用基本词典进行分词,保存切分开的基本词序列。

在索引创建过程中,对文档正文进行两趟分词。首先是基于基本分词词典的常规中文分词,采用北京大学计算语言所的分词软件。分词执行中包括复杂的歧义处理过程。第二趟再对基本分词结果使用基于扩展词典的分词,这一分词过程的最小单位是基本词典里的词条,采用正向最大匹配分词算法。两次分词的结果都被选择作为索引词,在倒排文件的创建中都被放入倒排索引词典,这一方法即混合索引。例如,基本词典有“国家”、“图书馆”两个基本词条,无“国家图书馆”;系统通过识别,发现“国家图书馆”极为可能是一个词语,于是把它加入到扩展词典。对文档中出现的“…国家图书馆…”字串,第一趟基本分词步骤把它切分为“国家”和“图书馆”两个基本词条,第二趟扩展分词再把它切分为“国家图书馆”,最终索引词包括“国家”“图书馆”和“/2 国家图书馆”这样三个单位。扩展分词结果使用转义符“/”标识,转义符后紧接扩展词包含的基本分词词条个数,用于查询时位置关系的计算。

混合索引的检索过程对用户输入的查询串执行同样的两趟分词。首先是基本分词,第二趟再对基本分词结果使用扩展分词。根据扩展分词结果词条包含的基本分词词条个数,标记被扩展分词结果覆盖的基本词条,它们在查询执行过程中无需处理。如上例,当用户输入查询“国家图书馆”,经过两趟分词,被切分为:“国家”、“图书馆”、“/2 国家图书馆”。其中前两个基本词条被第三个扩展词条覆盖,查询执行中只需直接读取索引词“/2 国家图书馆”对应的倒排项数据,即可完成查询执行过程。相对于分别读取“国家”和“图书馆”的倒排项数据,然后按其中的位置数据验证短语关系的方法,使用混合索引大大提高了检索效率。在混合索引条件下,当用户查询“图书馆”时,检索将按正常的查询过程执行,混合索引也不会降低系统的查全率。

与文献(Bahle et al. 2002)的短语索引相比,混合索引使用统一的倒排索引词典,没有额外的二级索引词典访问开销;并且混合索引不限制扩展词条为两个基本词条长,可以索引更长的短语,更加灵活。与词索引+Bi-gram 索引相比,混合索引使用了未登录词的识别技术,可以有效控制倒排索引词典规模,避免了 Bi-gram 词典膨胀的问题。混合索引也是索引结构的规模与检索效率间的一种折中。一方面,文档中的词被重复索引导致索引结构增大,占用更多的存储资源,另一方面,这些增加的

索引,使得更多可能形成词语或短语的字串被索引,可以大大提高对它们的检索效率。实际环境中,系统存储开销相对于检索效率不是那么重要,所以混合索引成为一项可用的技术。

## 二、混合索引的实现

混合索引的实现主要包括未登录词识别、扩展词典组织与分词两个部分。除了两趟分词和扩展词对基本词条的覆盖处理外,索引系统的创建过程和检索过程同一般的索引实现没有区别。

### 1. 未登录词的识别

目前,从语料库中自动识别或者学习词典未登录新词,特别是面向领域的专业词汇以及人名、地名、机构名等专有名词等方面,已经有了大量的研究工作和实用的技术。对文本数据常规的未登录词识别算法(刘开瑛 2000)一般包括如下步骤。

1) 提取  $n$  元组:使用基本词典,将文本进行部分分词,从部分分词结果中提取  $n$  元组,即包含  $n$  个相邻基本词条的字串。一般  $n$  元组的规模很大,不利于后续处理,常通过设定  $n$  元组的提取规则加以限制。例如,可以设置如下规则:对单字,只有连续出现的单字才能生成  $n$  元组;形成新词的  $n$  元组必须包含一个单字等。

2) 噪声剔除:删除那些包含低构词能力字的  $n$  元组,如常见的助词“的”、“得”,介词“在”、“把”等。对于这些字,由于数量少,可以人工收集包含它们的词。

3) 剔除  $n$  元重叠:把那些在  $n$  取不同值情况下重复被提取的  $n$  元组剔除。

4) 剩下的  $n$  元组按出现频次降序排列,为识别结果。

未登录词可以从网页文档集合与用户查询日志两种基本语料数据中得到。网页文档集合规模很大,用简单的  $n$  元组提取方式执行完整的未登录词识别算法会产生数量巨大的候选元组,难以处理。实际实现时,通过加入一些提取规则限制提取范围,使得在识别效率和识别效果间取得平衡。第一个规则的基本出发点是,常被用户查询的词或短语应该在文档集合中比较重要,而重要的词往往会在网页中的一些特殊位置和标签中间出现,如标题、H 标签、加重的文字等。网页中这些文字首先被选择执行  $n$  元组提取。第二个规则,网页和普通文本不同,网页中通常会有大量已经被自然分隔的较短的字串,分隔包括标点,网页标签,如导航链接上的文字、选择框中的文字、表格等。这些较短的字串极可能就是一个词,被选取执行  $n$  元提取。第三个规则是从网页原文摘要文字中提取  $n$  元组。天网搜索引擎使用了文档模型技术(Zhang et al. 2004)对搜集系统得到的网页进行预处理,提取了网页正文的摘要。摘要文本是正文中最重要的内容,而且长度相对于正文全文小很多。通过上述三个规则,可以有效控制未登录词识别执行的数据规模,提高识别效率,而且提取的是整个文档集合中重要的文本内容,对识别效果没有太大损失。

从用户查询日志中识别未登录词,除了使用基本的识别方法外,还可以利用查询



日志数据的特点。例如,直接从网页文档语料库中学习新词需要较复杂的处理算法,处理数据量大,而日志文件中用户的查询词通常比较简短,几乎没有完整的句子,只需做一些简单的处理,即可方便地学习新词。而且,如果用户查询词在词典内没有收录,很有可能就是潜在的新词,所以学习新词的准确率也比较高。搜索引擎日志中用户查询词在频度分布上是高度集中的。通过对天网用户查询日志的统计分析表明,前5%的高频词占据了64%以上的总查询次数;前20%的查询词占据了83%以上的总查询次数。我们可以利用查询词的频度分布特性来大量地减少 $n$ 元组提取过程中处理数据量。而且从提高系统检索效率的角度,对长期保持高频度的用户查询可以直接加入到扩展词典里,越过识别过程。

## 2. 扩展词典组织与分词

识别出的新词保存在扩展词典中。基于扩展词典的分词实际上是一个对基本分词结果序列在扩展词典上的最长匹配查找过程,即输入基本分词结果序列,找到序列中在扩展词典里的所有最长的匹配词条。为了高效实现对分词的支持,先把每个新词字串用基本词典进行分词,转换为一个基本词条的序列。再使用一个散列查找表把基本词条的字串转换为连续的整数编码。扩展词典保存每个词条中基本词条的整数编码,这相当于一个由数字组成的 $n$ 元组的集合: $D = \{(t_1, t_2, \dots, t_n)\}, 2 \leq n \leq 6$ 。

系统中 $n$ 取2到6,式中 $t$ 为基本词条编码, $D$ 表示扩展词典。在扩展词典保存时,各元组按第 $k(1 \leq k \leq 6)$ 个元素的 $t$ 值进行基数排序。再把排序结果每一层上相同 $t$ 值的节点合并,转换为树结构保存。示例如图8-20所示。

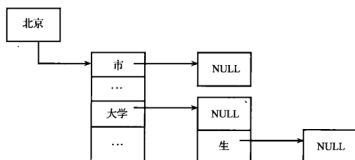


图 8-20 扩展词典树结构示例

在图8-20中,树节点保存实际上是基本词条的编码,且每一层的兄弟节点按节点内编码数值有序。基于这样数据结构的扩展词典匹配查找算法如图8-21所示。

输入:基本分词结果词条序列( $t_1, t_2, \dots, t_i, \dots$ )

输出:最长匹配扩展词条

算法:

```

1.  init-scoreboard();初始化匹配任务表
2.  while(ti! = EOF){
3.      code=get-code(ti);从编码散列表中取得  $t_i$  的编码
4.      for each task in scoreboard{
5.          ret=search-token(code);检测匹配任务追加一个词,是否结束?
6.          if{ret=NULL.}{
7.              clear-task
8.              add-hit           ;得到一个匹配
9.          }
10.         else update-task      ;根据检测结果更新匹配任务状态
11.     }
12.     check-hit      检查匹配结果,输出。
13. }
```

图 8-21 扩展词典匹配查找算法

## 第六节 倒排文件缓存机制

缓存技术是提高系统性能和可扩展性的一种重要手段,在计算机各个应用领域都有广泛的应用。如何有效地在搜索引擎检索服务系统中使用缓存技术也是近年来学术界广泛关注的问题。

缓存技术的有效性建立在被缓存对象访问序列存在的局部性特征上。与操作系统内存管理、数据库系统和 Web 代理缓存这些领域大量的研究相比,搜索引擎检索系统上的缓存研究相对较少。它们之间有其共性,但由于被缓存对象特征和对象访问模式的差异,又各自具有自己的特点。搜索引擎检索系统中通常被研究的缓存对象可分为三种,即查询结果、布尔操作的中间结果及倒排文件。文献(Xie et al. 2002, Wang et al. 2001)详细分析了搜索引擎用户查询日志,发现用户查询具有很强的局部性,提出了缓存查询结果的可行性。在文献(Wang et al. 2001, Markatos 2001, Saraiva et al. 2001)中,进一步研究了缓存替换算法、缓存大小等因素对系统性能的影响。天网在(Wang et al. 2001)的基础上实现了查询结果缓存,有效地提高了系统性能。文献(Chidlovskii et al. 1999)提出语义缓存,把布尔查询的中间结果作为缓存对象,并利用查询结果间的语义关系加速后续查询的执行。这种方法可以充分利用不同查询之间的相关性提高缓存命中率,缺点是限制在布尔查询上,可能影响结果相关性排序。第三种是倒排文件的缓存,用户查询经过查询器执行,转换为对倒排文件数据的访问序列,这些数据也可以作为缓存对

象。文献(Jonsson et al. 1998)研究了 IR 背景下用户交互式查询的倒排文件缓存与查询执行结合的方法,文献(Saraiva et al. 2001)研究了一个实际搜索引擎(TodoBR)中的倒排文件缓存对系统效率的影响。

下面基于天网的实际运行数据,重点讨论倒排文件缓存的优化设计。与(Saraiva et al. 2001)相比一个差异在于它使用的是过滤向空间模型查询处理技术(Persin et al. 1996),而天网的查询处理考虑查询词位置邻近关系,使用带位置数据的倒排索引,并使用索引压缩和块随机访问技术提高性能(Navarro et al. 2000)。这种查询处理技术的不同,导致所产生的访问倒排文件数据序列性质的差异。文献中对倒排文件缓存的研究,基本以固定大小的页面为单位,忽略了倒排文件访问数据是变长这一特点;并且缺乏替换策略、数据组织对缓存效率影响的分析。本节就如下问题展开讨论。

- 1) 缓存性能评估的指标如何选取。
- 2) 倒排文件缓存与操作系统的文件缓存相比是否有优势。
- 3) 倒排文件的数据组织方式对缓存效率及系统性能的影响如何。

## 一、倒排文件缓存

### 1. 体系结构

天网检索系统采用分布式体系结构,按文档划分的方式组织数据到多个索引服务节点,它们独立的并行执行用户查询,把各自检索结果提交给查询服务器汇总返回给用户。各级缓存的位置如图 8-22 所示。

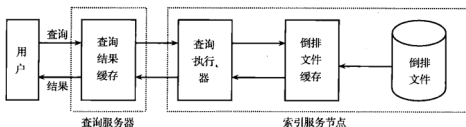


图 8-22 搜索引擎检索系统缓存结构

倒排文件缓存位于索引服务节点上,对查询执行器在执行用户查询过程中访问的倒排文件数据进行缓存。大量统计研究表明用户查询词序列具有良好的局部性,可以预期查询执行器发出的读取这些查询词倒排数据序列也具有同样的性质,这是人们研究倒排文件缓存的基本出发点。

在搜索引擎应用环境下,用户提交的查询中包含查询词个数通常很少,而词间的位置邻近关系对结果排序十分重要。与(Saraiva et al. 2001)的过滤向空间模型查询处理技术不同,天网使用带位置数据的全文倒排索引,对多个词的用户查询计算

邻近权值。查询执行器访问倒排文件的数据分为两类：一是查询词对应的倒排表中的文档编号和文档内权值数据，称为文档数据；另一部分是查询词对应的出现在每篇文档中的位置数据，称为位置数据。执行过程中，各个查询词按倒置文档频率降序处理，先读取文档数据，执行文档集合的布尔运算（通常搜索引擎默认为 AND），得到一个小的结果集合，同时使用文档内权值数据计算每个结果文档对查询的相关性权值；再读取对应的位置数据，对结果集合进行邻近权值排序。通过索引压缩技术，再结合对高频词使用位图记录文档数据，可以有效控制文档数据的长度。一般情况下，位置数据总量是文档数据量的 3~4 倍，查询执行中不必全部读取，通过随机访问的倒排索引组织技术(Navarro et al. 2000)，可以减少数据读取量。

天网查询执行器读取倒排文件的数据序列包括文档数据序列和位置数据序列。每次读取的数据长度不固定，但基本可以在一次磁盘系统 I/O 操作下完成（不考虑文件系统开销和数据组织碎片的条件下），这样的数据序列总称为 I/O 序列，它的项数代表磁盘系统执行的 I/O 次数。另外，以固定页面为单位，可以把 I/O 序列转换为页面访问序列，称为 PAGE 序列，它的项数对应磁盘系统实际读取数据总量，这可以衡量磁盘系统的带宽使用效率。搜索引擎的查询执行属于磁盘密集型应用，和数据库系统的事务处理应用类似，瓶颈在于磁盘系统每秒能够执行 I/O 次数的能力，即其 IOPS 参数；但平均每次访问的数据量比数据库事务处理要大，磁盘系统带宽参数也不能忽略。从实际系统的性能角度分析，与考察 PAGE 序列相比，考察 I/O 序列对缓存系统的性能评估更为有意义。

## 2. 负载数据

采用踪迹驱动(trace driven)的方法来研究倒排文件缓存的性质。采用天网 2002 年 11 月的用户查询日志，在北大燕穹提供的数据产品中的编号为 YQ-QUERYLOG.021203(InfoMall 2004)。查询日志记录了用户查询是否被天网的查询结果缓存命中。把被结果缓存命中的查询剔除，就得到实际到达索引服务节点图 8-23 所示的查询序列。

同时，还需要形成一个该查询序列所针对的文档集合。为此，从天网搜集的网页集合中随机抽取一批网页，建立索引，修改查询程序，把访问倒排文件的每次操作记录到日志文件，内容包括访问数据的索引词编号、文件偏移、数据长度、访问类型。其中访问类型采用位置数据块的编号。取经过过滤的查询序列中连续的 10 万个查询，送入查询程序执行，记录得到此查询序列在天网查询执行器下的倒排文件访问序列。根据访问日志中的索引词编号、访问类型与数据长度可以得到 I/O 序列，根据文件偏移和数据长度和指定的页面大小(如 4KB)，可以得到 PAGE 序列。

最后，为了更有效地处理数据，把两个序列中的对象标识(I/O 序列中是索引词编号与访问类型，PAGE 序列是页面编号)转换为从 0 开始的连续整数。数据集的统计信息如表 8-5 所示。

表 8-5 数据集基本统计信息

名 称	数 值	名 称	数 值
用户总查询数	7 341 383	I/O 序列长度	1 887 198
结果缓存未命中个数	3 522 968	I/O 序列唯一对象数	112 145
文档总数	2 603 035	PAGE 序列长度	20 808 025
文档数据原始大小	30.18 GB	PAGE 序列唯一对象数	965 929
倒排文件大小	5.77 GB		

## 二、负载特性

这一部分分析负载数据的性质,讨论它们对倒排文件缓存和缓存替换算法的影响。

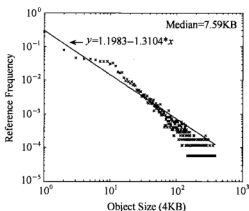


图 8-23 文档数据访问对象大小分布

### 1. I/O 序列对象大小

I/O 序列中的对象大小不同,其中由位置数据访问产生的部分是固定长度(32KB),而对文档数据访问产生的对象大小分布很不均匀,以 4KB 为单位对其分布统计如图 8-23 所示。其中值为 7.59KB,79% 的请求对象长度在 64KB 以下,同时也有少数较大的数据访问。

有效的缓存替换算法需要考虑对象的大小。对大量的小数据对象优先缓存,可以提高缓存的命中率,而对大对象优先缓存可以提高缓存的字节命中率。因为 I/O 序列反映的是系统 I/O 请求的次数,所以缓存命中率更为重要。在考虑缓存替换策略时,偏向小对象的方法预计可以获得更好的性能。

### 2. 序列中对象的频度分布

对象被访问的频率是缓存设计的一个重要因素。如果序列中对象访问频率分布非常不均匀,则需要考虑两个问题:一是缓存少数高频对象可以提高性能,另一个是不区分出大量低频对象将降低性能。实际上对象的访问频率和访问的时间局部性是相关的,可以推导出高频的访问对象也会具有较高的访问时间局部性(Jin et al. 2000)。

I/O 序列和 PAGE 序列的访问频度对其访问频度的序号的分布如图 8-24 所示。在计算数据点序号时,对同频度的数据使用最后一个数据点的序号,这样图中曲线平

滑,没有尾部数据点堆积,便于分析和比较。总体看,两个序列都存在访问频度分布不均匀的现象,但和通常的 Zipf's 分布相比,这种差异还算是很平缓,可以预期频率是倒排文件缓存替换算法应该考虑的一个因素。但只考虑频率的替换算法,如 LFU,效果不会很好。两者间,I/O 序列的频率特性比 PAGE 序列更有利于缓存应用。

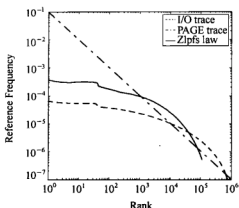


图 8-24 I/O 与 PAGE 序列序号-频度分布

### 3. 序列中对象的时间间隔分布

序列的时间局部性可以从序列中对同一个对象的两次连续访问的时间间隔分布来考察。使用访问在序列中的位置间隔,而不使用绝对时间,可以屏蔽用户查询密度在各个时间段内的周期性对分析的影响(Jin et al. 2000)。

I/O 序列和 PAGE 序列的时间间隔分布如图 8-25 所示。由于直接的时间间隔分布十分散乱,图中的处理是把距离数据按 2000 为单位分组,表现的是各组的频度。可以看到对数坐标下,序列的时间间隔分布接近直线,说明具有良好的时间局部性。I/O 序列的斜率为 1.039, PAGE 序列为 0.764,表明在同样的缓存大小比例下,I/O 序列可以预期得到比 PAGE 序列更高的缓存命中率。较强的时间局部性有利于缓存设计,对象访问的时新性(freshness)是替换算法需要考虑的一个重要因素。

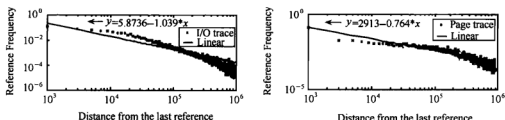


图 8-25 I/O 与 PAGE 序列时间间隔分布

### 4. 序列的重复模式

序列的空间局部性是指序列中固定模式的重复,可以通过原始序列和随机排列处理后的序列中的唯一的定长串的个数来说明(Almeida et al. 1996)。空间局部性也是缓存设计需要考虑的因素。

取 I/O 序列和 PAGE 序列前 10 万个数据,处理得到其中长度为 1~9 的连续

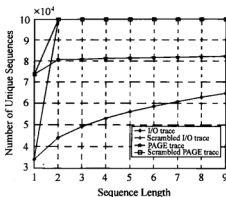


图 8-26 I/O 和 PAGE 序列中唯一模式串

串,统计唯一串的个数。再把序列进行随机重排,重复统计。得到序列中指定长度的唯一串的个数如图 8-26 所示。随机排列破坏了序列中的重复模式,即破坏了序列的空间局部特性。图中随着串长度的增加,唯一串的个数也增加。随机排列的序列增加速度最快,其空间局部性最差,I/O 序列增加得最为平缓,其空间局部性较强,PAGE 序列次之。

### 三、缓存策略的选择

现代操作系统的文件系统通常都提供 I/O 数据的缓存功能,通常以页为单位。也就是说,如果在应用层不安排缓存,应用中发生的 I/O 操作物理上也都在内存发生。文献(彭波 2004b)通过四组缓存仿真实验,验证了倒排文件缓存经过优化设计,可以比文件系统缓存性能更好。具体的方法可以通过缓存变长的 I/O 序列对象,选择性能更好的 GD-SIZE1 替换算法,从优化磁盘系统 I/O 次数的角度来提高系统性能;也可以通过选取大的页面作为访问倒排文件的单位,从优化磁盘系统带宽利用率的角度提高系统性能。最后按页面对齐的方式组织倒排文件可以进一步优化缓存和系统性能。而这一组织方式下,可以直接把倒排文件页面存放在磁盘设备上,通过直接的设备访问接口,越过文件系统 read/write 调用来访问倒排文件数据。再加上优化设计的倒排文件缓存,使得索引服务的性能得到进一步的提高。

## 第七节 小 结

本章第二、三、四节很多内容直接来自单栋栋同学的研究报告。本章第一节通过分析天网检索子系统的设计与实现,概述了检索系统所要关心的若干基本技术。检索系统的设计目标围绕检索效果和检索效率两个方面展开。系统通过一个集成框架把多种技术融合到一起,包括中文文本自动分类技术、中文信息提取技术等,以求能不断提高检索质量。

首先介绍适于查询的通常倒排索引结构,然后介绍一种可以提高短语查询速度的查询结构——位置倒排索引。给出了位置索引的详细结构及其在这种结构上的查询处理算法;给出了缓存敏感的查找表算法,该算法用于把出现短语的位置通过文档边界数组转成对应文档号。

接下来四节介绍的技术是为了减少存储倒排索引的空间,并提升查询处理的速度。

针对倒排索引压缩技术,介绍了一些常用的倒排索引的压缩算法,详细的综述和对比讨论如何高效的对倒排索引中文档号序列,词频序列及位置序列进行压缩,以及使用哪些压缩算法比较合适。

索引剪枝技术主要是从减少倒排索引的大小或者查询处理时尽量少的处理数据这两方面来提升查询的处理速度。

混合索引技术采用了一种基于未登录词自动识别技术来实现的。实践表明,这一方法可以有效提高搜索引擎检索效率。针对搜索引擎中倒排文件缓存技术。通过分析数据访问序列的局部性特征,以及基于真实数据的缓存仿真实验(彭波 2004b),探讨了倒排文件缓存优化设计中的性能指标选择问题、替换算法、页面大小和倒排文件组织方式等对缓存性能的影响,得到如下结论:①通过缓存变长的 I/O 序列对象,采用 GD—SIZE1 替换算法,可以明显减少磁盘系统 I/O 访问的次数;②通过按页面对齐方式组织倒排文件,选取大的页面作为访问倒排文件的单位,可以使磁盘系统带宽利用率得到优化。

本部分涉及的源程序请参看网址 <http://sewm.pku.edu.cn/src/paradise> 中的分析和索引(analysis & index)模块。



## 第九章 相关排序与系统质量评估

传统上,人们将信息检索系统返回结果的排序称为“相关排序”(relevance ranking),隐含其中各条目的顺序反映了结果和查询的相关程度。在搜索引擎的情形,人们也这么讲,但内涵其实是有了差别。一方面,搜索引擎维护的内容十分繁杂且不规范,不像传统的图书、文献等有很好的分类体系管理。另一方面,搜索引擎面对的用户背景广阔,层次多样,不像传统的图书馆所面对的用户通常有相对比较整齐的用户群。因此,搜索引擎要给出的不是一个狭义的相关序,而是某种反映多种因素的综合统计优先序。检索质量评估的目标是对不同搜索引擎系统的检索质量评估其相对优劣次序。本章的第一节介绍传统的相关排序技术;第二节分析网络环境下影响排序的若干新的因素,并讨论如何利用 Web 间的链接关系进行相关度排序;第三节给出了考虑这些因素后的一个结果排序的具体实现方案,第四节介绍搜索引擎系统质量评估的一般技术与方法。

### 第一节 传统 IR 的相关排序技术

给定某个文档集合  $D$ , 大小为  $M$ ; 设两篇文档  $d_1, d_2 \in D$ , 一个查询  $q$ 。用什么样的标准来讲“ $d_1$  与  $d_2$  相比, 前者和  $q$  更相关”? 这方面最经典、最有影响的工作是 Gerald Salton 等在 30 多年前提出的“向量空间模型”(Vector Space Model, VSM) (Salton et al. 1968, Salton 1971)。该模型的基础是如下假设: 文档  $d$  和查询  $q$  的相关性可以由它们包含的共有词汇情况来刻画。

这样, 文档  $d$  和查询  $q$  就都被简化成词汇的集合(多重集)。不失一般性, 令

$$\sum = \{t_1, t_2, \dots, t_N\}$$

为一个词典,  $t_i$  为词项,  $N$  为它的规模, 则

$$d = \langle t_1^{m_1}, t_2^{m_2}, \dots, t_N^{m_N} \rangle$$

$$q = \langle t_1^{n_1}, t_2^{n_2}, \dots, t_N^{n_N} \rangle$$

其中,  $m_i, n_i (i=1, 2, \dots, N)$  表示相应词项出现的次数, 即词频 TF; 如果次数为零, 则表示该词项在文档或查询中没有出现。在实际应用中, 人们通常去掉  $t_i$ , 而直接用  $m_i, n_i$  来表示  $d$  和  $q$ 。

$d$  和  $q$  相关程度的评价就以这样两个向量的某种“相近程度”为基础, 这其中, 有一些细节的变化。

1) 上述表示中, 词项在文档和查询中出现的次数(词频)是一个基本量, 称为“词

频”模型。在实用中用规格化表示(以一篇文档为例)。

$$d = \langle w_1, w_2, \dots, w_N \rangle \text{ 其中, } w_i = \frac{m_i}{\sum m_j}$$

这里,  $w_i$  也称为词频, 称这种方式为用词频来表示词项在文档或查询中的权重。查询  $q$  也有同样的表达形式。

2) 在许多情况下, 为了简便起见  $m_i, n_i$  只在集合  $\{0, 1\}$  中取值, 表示词项的出现与否, 不关心其次数; 此时的模型称为“二元模型”。

3) 若一个词项  $t_i$  在许多文档中都有出现(如“我们”、“大家”等), 尽管它可能在文档内部出现的频度也很高, 它对于不同文档的区分能力就不会很强, 因此它的权重应该相对较小。将这种观念刻画出来, 引入词项的文档频率 DF 的概念。用  $k_i$  表示词项  $t_i$  在文档集合  $D$  中涉及的文档个数,  $M$  表示集合  $D$  的大小, 则文档频率为  $DF(t_i) = \frac{k_i}{M}$ 。

我们需要的是和 DF 成反比的一个量, 称为倒置文档频率 IDF, 常用的一种定义为  $IDF_i = \lg\left(\frac{M}{k_i}\right)$ 。这样结合词频, 就有了经典的 TF \* IDF 词项权重的设计

$$w_i = TF_i \times IDF_i = \frac{m_i}{\sum m_j} \times \lg\left(\frac{M}{k_i}\right)$$

给定某种权重的定量设计, 求文档和查询的相关性就变成了求  $d$  和  $q$  向量的某种距离, 最常用的是余弦(cos)距离

$$\cos(d, q) = \frac{\sum w_i^{d_i} \cdot w_i^{q_i}}{|q| \times |d|}$$

上述这些理论, 源于传统信息检索领域, 针对的是普通文本。这样一种理论虽然从根本上看起来比较粗糙(将文本近似成一个词项集合, 完全忽略语法和语义), 但几十年来在大量真实语料评估的驱动下, 其不断完善的实践在实践中得到普遍认可<sup>①</sup>。

从具体实现的角度看, 这样一种理论在倒排文件的数据结构上能够很容易得以实现。给定文档集合  $D$  和词典  $\Sigma$ , 对  $D$  中的每一个  $d$  得到其权重表示(那些  $w_i$ )是预处理的工作, 同时自然也得到了所需的  $M, N$  等。

现在的问题是, 我们需要得到网页(用  $p$  来表示)和查询  $q$  的相关性。最初, 一种简单的方法就是用普通文本作为网页的近似, 即,  $p \approx d$ , 只考虑网页中那些用户可见的文字部分, 忽略标记和超链等内容, 于是上面的理论和实践都可以马上照搬。但人们很快发现这有很大问题, 其原因在于传统 IR 方法的成功有两个重要的内在假设:

1) 被索引的信息本身有很高的质量, 至少在信息的组织和内容上有着比较高的质量。在 Web 出现以前, 传统的 IR 之所以能够行之有效至少在部分是依赖这一

① 这给我们一种启示: 在应用性很强的领域, 理论基础粗糙的先天不足可能由精细的实现技术来弥补。

点的。很多的 IR 产品一般都是针对一个特定的领域,如法律信息、医疗信息、环保信息等等。这样它们可以针对这个领域进行算法的优化,同时,也避免了对一词多义的处理。

2) 检索信息的用户有一定的相关技能和知识。也就是说,当用户面对一个很大的信息源时,他知道通过什么样的手段去提高检索的准确率,但同时又不降低系统的查全率。与此相对应的,传统的 IR 系统总是提供一套相当复杂的检索语法来满足用户的不同要求。

然而,这些假设在 Web 上都已不再成立:

1) Web 上网页的质量参差不齐,大量的网页组织性、结构性比较差。同时,Web 又是一个无所不包的载体,它涉及政治、经济、教育、生活等各个层面。这使得 IR 中的很多技术都没有了施展才能的余地。另外,网络上充斥着很多没有任何意义的网页,很多镜像的网页,如果不采取相应的技术处理,将会在很大程度上影响检索质量。

2) 大部分检索用户是没有任何经验的。他们经常只输入一个或者两个检索词来检索需要的网页,但会得到大量的返回结果,很难达到满意的程度。很少有用户愿意使用逻辑运算来提高检索的质量。即使这样,在不少用户的输入表达中,依然存在各种各样的问题<sup>①</sup>。

基于此,人们发现原有的 IR 技术已经不能适应 Web 的发展,必须改进原有的 IR 相应技术,研究新的适合 Web 的技术和算法,提高 Web 检索的质量。

## 第二节 链接分析与相关排序

尽管 Web 页面的情况比传统 IR 面对的情况要复杂许多,但其中的复杂性也给我们带来了新的机会,主要体现在两个方面。首先可以利用网页间的链接关系进行链接分析,量化网页信息;其次,在 Web 查询模式下产生了许多新的信息可资利用,如 Web 用户行为信息等。

### 一、链接分析

从开发利用的角度看,网页和普通文本的不同主要反映在两个方面:HTML 标签和网页之间的超链接。

1) HTML 设计有丰富的标签,是网页作者用于将网页的不同部分以不同的形式呈现给用户的手段,包括文字的布局,字体、字号的变化等,主要追求的是视觉效果。因此,标签能给我们提示其中文字的重要程度。例如,常识告诉我们,在同一篇文章中,比较大的字体往往是作者比较强调的内容;而在一版(以区别“一篇”,如同报纸)内容分块、且有一定布局的文字上,放在前面和中间的应该是作者比较强调的,等

<sup>①</sup> 例如,在天网日志统计中发现“搜索”这个词曾多次出现,反映了不少用户还不知道该如何用搜索引擎。

等。许多著名搜索引擎在网页的预处理阶段记录了这些信息,并用于结果排序。如 Alta Vista, Inktomi, Excite, Infoseek 等。

2) 链接反映的是网页之间形成的“参考”、“引用”和“推荐”关系。可以合理地假设,若一篇网页被较多的其他网页链接,则它相对较被人关注,其内容应该是较重要或者较有用。因此,可以认为一个网页的“入度”(指向它的网页的个数)是衡量它重要程度的一种有意义的指标。这和科技论文的情况类似,被引用较多的就是较好的文章。同时,人们注意到,网页的“出度”(从它连出的超链个数)对分析网上信息的状况也很有意义的,因此可以考虑同时用两个指标来衡量网页。这些想法即是斯坦福大学 Google 研究小组和 IBM 公司的 Clever 系统开发小组几乎在同一时间分别提出著名的 PageRank 技术和 HITS 技术的基础。

可以用一种“随机冲浪”模型来作为 PageRank 的理论基础,该模型描述网络用户对网页的访问行为,假设如下:

- 1) 用户随机的选择一个网页作为上网的起始网页;
- 2) 看完这个网页后,从该网页内所含的超链内随机的选择一个页面继续进行浏览;
- 3) 沿着超链前进了一定数目的网页后,用户对这个主题感到厌倦,重新随机选择一个网页进行浏览,如此反复。

按照以上的用户行为模型,每个网页可能被访问到的次数越多就越重要,这样的“可能被访问的次数”也就定义为网页的权值,PageRank 值。如何计算这个权值呢? PageRank 采用以下公式进行计算:

$$W_j = (1-d) + d \sum_{i=1, i \neq j}^N l_{i,j} \frac{W_i}{n_i}$$

其中,  $W_j$  代表第  $j$  个网页的权值;  $l_{ij}$  只取 0、1 值,代表从网页  $i$  到网页  $j$  是否存在链接;  $n_i$  代表网页  $i$  有多少个连向其他网页的链接;  $d$  代表“随机冲浪”中沿着链接访问网页的平均次数。选择合适的初始数值,递归的使用上述公式,即可得到理想的网页权值。

IBM 研究院 Clever 系统中的相应技术称为 HITS(Hyperlink-Induced Topic Search)。Clever 描述两种类型的网页。

- 权威型(Authority)网页:对于一个特定的检索,该网页提供最好的相关信息;
- 目录型(Hub)网页:该网页提供很多指向其他高质量权威型网页的超链。

进而在每个网页上定义“目录型权值”和“权威型权值”两个参数。当遇到一个检索时, Clever 先利用检索的关键词得到一个网页的根集合,如从搜索引擎返回结果取前 200 个网页;然后根据这个集合在整个网页有向图中的位置来扩展这个根集合。具体办法是,将被链接(包括链出和链入)的网页加入到这个根集合中,形成一个新的集合;依据指定的网页规模作扩展,如使根集合扩展到一个包含 1000~5000 个网页的集合。

在得到这个集合后,就开始计算集合中每个网页的目录型权值和权威型权值。Clever 的做法是采用目录型网页和权威型网页相互评价的办法进行递归计算。对于一个网页  $p$ ,用  $x_p$  来表示网页  $p$  的权威型权值,用  $y_p$  来表示它的目录型权值,并且用如下公式进行计算:

$$x_p = \sum_{q \text{ such that } q \rightarrow p} y_q$$

$$y_p = \sum_{q \text{ such that } p \rightarrow q} x_q$$

这样的递归式也容易用矩阵方法表示。令所有选出来的网页都进行标号,得到所有网页的编号集  $\{1, 2, \dots, n\}$ 。令相邻矩阵  $A$  为一个  $n \times n$  的矩阵,如果存在一个从网页  $i$  链接到网页  $j$  的超链,就令矩阵中的第  $(i, j)$  个元素置为 1,其他各项置为 0。同时,将所有网页的权威型权值  $x$  和目录型权值  $y$  都表示成向量形式  $x = (x_1, x_2, \dots, x_n)$ ,  $y = (y_1, y_2, \dots, y_n)$ 。由此可以得到计算  $x$  和  $y$  的简单矩阵公式:  $y = Ax$ ,  $x = A^T y$ , 其中  $A^T$  是  $A$  的转置矩阵。进一步,有

$$x = A^T y = A^T A x = (A^T A) x$$

$$y = A x = A A^T y = (A A^T) y$$

经过一定次数的递归运算后,会得到集合中每个网页的权威型权值和目录型权值。按照这两个不同的权值,分别取出前  $k$  个返回给用户。根据 Clever 系统自己的测试数据,对于返回给用户的前 10 个检索结果, Clever 系统在 50% 的情况下获得了高于 Yahoo! 和 AltaVista 的用户评价。

通过上面的分析,发现这两种方法有很多相似之处。它们都利用了网页和超链组成的有向图,根据相互链接的关系进行递归的运算。但是,两者又有很大的区别,主要在于运算的时机。Google 是在网页搜集告一段落时,离线的使用一定的算法计算每个网页的权值,在检索时只需要从数据库中取出这些数据即可,而不用做额外的运算,这样做的好处是检索的速度快,但丧失了检索时的灵活型。Clever 使用即时分析运算策略,每得到一个检索,它都要从数据库中找到相应的网页,同时提取出这些网页和链接构成的有向子图,再运算获得各个网页的相应链接权值。这种方法虽然灵活性强,并且更加精确,但在用户检索时进行如此大量的运算,检索效率显然不高。

## 二、Web 查询模式下的新信息

上述链接分析可以有效地计算网页的重要程度,但是带有明显的偏向,即不重视新出现的网页。新出现的网页,尽管可能很重要,但由于时间短,被链接的次数显然不可能很高,PageRank 的值就不会高。因此需要来补偿这个问题,人们注意到,除网页本身特性外,搜索引擎的应用环境和传统信息查询也有些不同,这可以从两个方面考虑。

### 1. 用户行为

和传统 IR 的用户群相比,虽然搜索引擎的用户群的经验少,但他们的数量却十分巨大。大型商业搜索引擎,如 Google、AltaVista、百度等,每天都有上 1000 万次的用户检索。通过对这些用户检索行为的统计分析,可以从中获取许多有用信息,这些信息可以大大提高搜索引擎检索结果的准确率,提高检索的质量。

Direct Hit 技术就是基于以上思想创立的。这项技术的主要特点是跟踪用户对检索结果的后继行为:哪些站点被用户选择浏览了?用户在这个站点上花费了多少时间?通过对这些数据的统计,搜索引擎就可以提高那些经常被用户选择,而且花了大量时间去浏览的站点的权值,降低那些不太被用户关心的站点的权值。对于新加入系统的网页,系统则先给它们一个缺省的权值,然后由用户来决定它们的重要性。另外,系统还可以利用以前的用户检索行为来对以后的相似检索进行优化,帮助用户尽快发现自己需要的信息。

这个技术的另一个优点就是可以对一个固定的用户的行为进行跟踪和统计,进而发现这个用户的喜好和对检索结果的期待,从而产生专门针对该用户的检索结果。这就是我们后面会提到的个性化检索。例如,一个做建筑材料的工程师,当他检索“windows”,他最大的可能是关注有关窗户的问题;一个计算机工程师,同样的检索则更关心微软的 Windows 产品。通过几次用户检索行为的跟踪学习,就可以获得这些信息,进而在以后的检索中,就对检索的输出结果进行针对用户的适应性调整。

Direct Hit 公司的 Gary Cullis 在搜索引擎 1999 年年会上将搜索引擎使用的四种技术:根据网页本身信息(Author)、根据超链链接关系(Other Author)、人工编辑产生的目录系统(Editor)和根据用户行为(User)进行了比较,如图 9-1 所示,得出根据用户行为的技术比其他几种技术无论在查准率和查全率上都有相当多的优势。

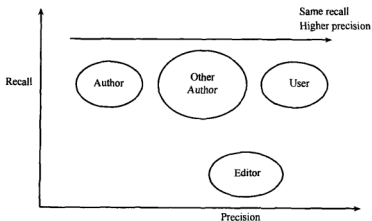


图 9-1 Inktomi 提供的几种搜索引擎技术的比较

## 2. 新词的产生

在本章的第一节中提到过词典  $\Sigma$ , 但没有强调它的重要性。事实上, 它是以关键词为查询表示的任何信息检索系统的基础, 中文尤其如此。在传统信息检索场合, 如图书馆, 信息资源相对稳定, 信息内容相对成熟, 词典也就相对稳定, 没有表现出突出的矛盾。网络环境下, 用户的信息需求很宽泛, 特别是“时代感”很强, 关注的内容与社会新闻和事件经常紧密相关, 因此在查询中常常会有那些时髦的新词, 如“大腕”、“美眉”之类。从我们前面介绍的倒排表工作原理可知, 如果词典中没有相应的词, 就不可能查到含有它们的网页(严格说是“不能有效地”查到它们, 原因见后)。因此, 获得新词, 将它们及时地加入词典中, 是维护运行搜索引擎的一个重要工作。下面是关于这一问题的论述。

简单地讲, 词典就是一个译码器, 它分配给词典中的每一个条目一个唯一的整型编码。前面已经论述过词典在预处理和查询服务两个阶段的必要性。在这里从系统设计的高度再分析一下词典的用途。将系统分为核心和外围应用, 图 9-2 显示了词典的地位, 它相当于系统内核和外围之间的一座桥梁。外围应用通常是和系统输入和输出(广义的)打交道的, 它们面临的数据千差万别。如果让核心直接处理这些形态各异的数据, 就会导致系统核心代码的急剧膨胀, 系统运行效率迅速降低。通过词典的处理, 将各种数据以统一的整型编码的形式交给系统内核, 使得系统内核的处理简单, 保证了系统的运行效率。

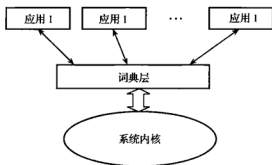


图 9-2 词典在系统中的地位

既然词典处于桥梁的地位, 那么词典本身的设计就十分的关键。天网系统采取了 Hash 表的方法来实现系统的词典。对于每一个输入数据  $D_{input}$ , 按照分布式搜集策略部分的分析, 可以将其对应到一个大整数。记 Hash 表的大小为  $h\_size$ , 对任意一个输入数据, 根据公式都可以获得它 Hash 表的入口地址

$$Addr = F_{key}(D_{input}) = \text{int}(D_{input}) \% h\_size$$

其中,  $F_{key}$  就是按照公式得到的 Hash 的散列函数。任何值域小于定义域的散列函

数,都不能保证没有冲突,这就要求散列函数造成冲突的概率尽量的小。通过选择适当的  $h$ -size 和散列函数,可以控制 Hash 的平均拉链长度。

对于散列函数  $F_{key}$ ,把 593 286 个项数据放入特征项大小为 100 万的 Hash 表中散列,得到的平均拉链长度为 1.32。这意味着,对任何一个数据  $D_{input}$  操作时,需要的 Hash 相关的操作为 1.32 次。其他的相关操作,如  $F_{key}$  的运算、Hash 表的增删改查,都是常数复杂度。因此,我们可以获得词典的运算复杂度为  $O(1)$ 。

在进行中文分词时,就需要依赖中文词典。复旦大学在它们的文档分类系统(黄菁萱等 1998)中,使用基于 Church 提出的计算互信息和  $\chi^2$  统计量的方法(Church et al. 1990)对文档进行专有词汇学习,所提取的专有词汇(5000 个)接近该系统使用的原始词典大小(11 000 个词条)的一半。结合关键词筛选的专有词汇学习技术使系统对开放语料的分类准确率提高了 15%。所以,按照需要不断扩大词典的容量是必须的。

如何扩大词典的容量?回顾对系统的分析,系统只有两个和外界数据的接口,即 Web 和用户检索。天网选择用户的检索进行学习,其理由如下:

首先,学习词汇是为了满足用户的检索需求,提高检索的质量。通过对用户检索数据的分析,从中学习新词,针对性强,更能提高检索的质量。

其次,从统计上来看,Web 上的数据和用户检索的字符串有着很大的差别。Web 网页中的中文大部分都是连写在一起的句子,以标点符号分开。而用户输入的检索字符串,根据我们的统计,大部分是词汇和词汇组成的短语。在网页中,面对一串连续的中文字符串,人们很难从中间提取出新词。而我们对用户的检索输入做一定的简单处理,即可以比较方便的学习到新的词汇。

可以通过如下步骤分析(图 9-3)。

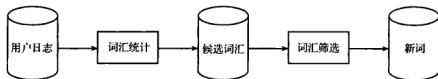


图 9-3 新词学习

词汇统计部分将用户的检索字符串进行一定的分析和筛选处理,并对通过筛选的词汇进行频率统计。①用户输入的检索有一部分是复杂的逻辑检索(大约 20%),应该首先将这些带有逻辑运算符的检索字符串转化为简单检索形式。②检索中有大量的英文检索和中英混合检索,这里处理的是中文新词学习,因此要将所有的英文词汇过滤。③前面已经提到,对于过长的中文字符串,它是一个词汇的可能性极小,为了提高学习的准确度,定义一个学习词汇的最大长度  $n$ ,把所有检索字符串串长大于  $n$  的过滤掉。④对这些合法的“可能新词”进行学习,统计出每个词汇的检索频率。



词汇筛选部分主要有两个步骤。首先进行词频筛选,将低频的检索排除在新词之外。众所周知,搜索引擎用户可以随便的检索任何内容,对于那些只有极少数人关心的生僻词汇,不用加入到词典中,因为词典的过度膨胀会在一定程度上降低系统运行的效率。另外,有很多的用户不能十分确定他们要检索的中文的正确拼写,或者在检索时不小心输入错误,如将“搜狐”输入成为“搜虎”。通过词频的筛选,可以在一定程度上解决这些问题。另外,统计得来的候选词汇,有一部分在词典中已经存在,需要将它们过滤;还有一部分是两个或多个合法词汇组成的短语,一样需要将它们过滤掉,如“计算机网络”。

学得的新词将和搜集端从网页中提取的特征项共同组成新的词典,为以后的搜集和特征项提取服务。通过新词学习技术,使得系统对于新词检索的准确率大大提高,表 9-1 给出了新词学习前后检索准确率的对比数据。

表 9-1 新词学习对检索准确率的影响

词 汇	学习前准确率/%	学习后准确率/%
克林顿	68	82
爱心社	49	85
宝洁	16	73
分布式	26	60
访美	58	76
非线性	66	76
编程	90	94
丰田杯	82	100
安阳	10	50
单片机	54	88

由于篇幅原因,这里仅列出了 10 个新词的检索结果。通过对 100 个新词的检索结果的统计,我们获得更新前的平均检索准确率为 49.7%,更新后的平均准确率为 78.8%,即搜索引擎系统对于新词的平均检索准确率提高了 58.6%。

### 第三节 相关排序的一种实现方案

将本章前述分析综合到一起,这一节给出相关排序的一种具体的实现方案。

一个网页是否重要,可以从其他网页上找出相应的线索。如果一个网页十分重要,那么会有大量的链接指向这个网页。因此,需要对一个还没有搜集的 URL 地址进行被链接次数的统计,以确定该 URL 获得的其他网页的评价,同时赋予其相应的权值  $W_i$ 。另外,可以根据人们日常在网上的访问,来获得一些有价值的网站,加入到配置文件中。当一个网页属于这些重要网站时,我们就赋予它另外一个权值  $W_i$ 。还

有就是网页的编码类型。作为一个主要为华人服务的搜索引擎,我们主要的关注点在中文信息,所以我们应该优先搜集那些中文网页。即便是中文,仍然有不同的编码类型。例如,中国内地主要以 GB 为主,港台地区则以 Big5 为主,在北美及欧洲地区还存在 HZ 编码。按照搜索引擎服务的用户群,应该给相应的网页赋以不同的优先搜集次序,在系统里,它体现为编码权值  $W_c$ 。

基于以上三个主要方面的考虑,得到一个 URL 的权值评价

$$W_i(p) = W_l(p) + W_s(p) + W_c(p)$$

这样,每个待搜集的网页都有自己的  $W_i$ ,超链选择程序根据这些权值,从中选出一个或一批权值最大的来搜集,即达到了我们期望的目的(雷鸣 2000)。

### 一、形成网页中词项的基本权重

前面提到了向量空间模型,但根据讨论,并不能够将它完全照搬到搜索引擎系统中来。网页信息和正文文本最重要的差别就是在网页中含有大量的 HTML 标签(tag)。因此,在天网中提出了一个改进的  $TF * IDF$ (Baeza-Yates et al. 1999, Church et al. 1990)算法用于检索和相关度评价算法。相对传统的 IR 而言,增加了对 HTML 标签和网页的可索引文本长度。可索引文本长度表示用户通过浏览器窗口看到的一个网页的文本内容长度。

考虑被 HTML 标签包围的一段文本内容,到底这些标签应该如何影响这段内容呢?天网将所有的标签分为两类:一类是影响文本权值的标签,如<FONT>、<H1>等;另一类是不影响文本权值的标签,如<IMG>、<FRAME>等。在此选择表9-2中的标签作为影响文本权值的标签。

表 9-2 影响权值的 HTML 标签

tag	$W_i(\text{tag})$	tag	$W_i(\text{tag})$
<TITLE>	40	<DT>	4
<CITE>	8	<LI>	4
<EM>	2	<UL>	4
<STRONG>	4	<A>	4
<B>	4	<FONT SIZE=7>	16
<I>	2	<FONT SIZE=6>	12
<BIG>	4	<FONT SIZE=5>	8
<H1>	12	<FONT SIZE=4>	4
<H2>	8	<FONT SIZE=3>	1
<H3>	4	<FONT SIZE=2>	1
<H4>	1	<FONT SIZE=1>	1
<H5>	1	<FONT SIZE=+1>	4

对于一个网页,首先给予该网页中的每个特征项一个缺省的权值  $W_0$ 。如果一个特征项还被其他的有权标签包围,这些标签的权值将会影响到这个特征项的权值。例如,“hello”在  $\langle \text{big} \rangle \langle \text{b} \rangle \text{hello} \langle / \text{b} \rangle \langle / \text{big} \rangle$  环境中的权值应该为

$$\text{WBT} = W_0 + W_i(\text{big}) + W_i(b)$$

通过此式,可以获得每个特征项在网页中每次出现的权值。假设特征项  $t$  在网页中出现  $n$  次,每次出现的权值分别为  $\text{WBT}_1, \text{WBT}_2, \dots, \text{WBT}_n$ , 就可以得到特征项  $t$  在整篇网页中的权值

$$\text{WBT}(t, p) = \sum_{i=1}^n \text{WBT}_i$$

对于一个网页,应用上式对每个特征项进行权值计算是公平的。但是,考虑到相同的特征项出现在不同的网页中,网页的长度越长,特征项可能获得的权值也就越高。所以,一个特征项的权值应该在某种程度上受到网页长度的影响。另外,为了区分高频词和低频词对网页的影响程度,我们沿用 IR 中的 IDF 项

$$\begin{aligned} \text{WB}'(t, p) &= \text{WBT}(t, p) \times \text{DS}(p) \times \text{IDF}(t) \\ &= \text{WBT}(t, p) \times \lg(S_{\max}/S(p)) \times \lg(N/T(t)) \end{aligned}$$

$S_{\max}$  表示最大的网页可索引文本大小;  $S(p)$  代表网页  $p$  的可索引文本大小;  $N$  代表被索引网页的总量;  $T(t)$  是包含特征项  $t$  的网页的数量。

通过以上的分析,可以看到一个特征项的权值由三部分组成:第一部分是考虑了 HTML 标签影响的绝对权值;第二部分是考虑网页大小对权值的影响;第三部分是特征项出现频率对权值的影响。

最后,对  $\text{WB}(k, p)$  进行归一化处理。其中  $\text{WB}_{\max}$  代表对于所有的  $k, p$  而言  $\text{WB}'(k, p)$  的最大值。

$$\text{WB}(k, p) = \frac{\text{WB}'(k, p)}{\text{WB}_{\max}}$$

$\text{WB}(k, p)$  将作为基本权值来参与相关度评价的运算。

## 二、利用链接的结构

网页之间的超链接是 Web 的基本特点,这也是从应用上区别现在的 Web 和以前的 Internet 最突出的特征。如果说 TCP/IP 协议组将上百万计算机无缝连接起来了,则 HTTP/HTML 协议组将上百亿信息(网页)无缝连接起来了。海量网页之间的相互链接形成了一个巨大的有向图(图 9-4),这个图的很多结构性特征既有趣,也有重要的意义。特别地,人们关心一个网页的入度。

在这部分,主要考虑 WWW 中超链的互链关系对一个网页权值的影响。Web 有两个基本的构成因素:网页和超链。如果将网页认为是节点,超链是有向边的话,就可以将整个网络抽象为一个巨大的有向图。从图中可以看到,每个网页的入度是不同的,称每个网页的入度为网页的链接命中数(Link Hit Number, LHN)。

为什么 LHN 应该影响一个网页的权值呢? 我们知道这些超链都是网页的编辑加入网页中的。之所以加入这些超链, 是他们认为这些超链是有价值的, 值得他们网页的浏览者去深入浏览。如果一个网页被大量的其他网页所链接(也就是说, 被大量的网页编辑推荐), 可以确定, 这个被链接的网页是相对重要的, 它们会对上网浏览的用户有更大的帮助。

所有的链接都应当按照如上所述的方法考虑吗? 经过分析, 天网将超链分为两类: 链接向本网站内部网页的超链(自我推荐)和链接向其他网站上的网页的超链(他人推荐)。忽略第一类链接, 理由如下。

1) 通过统计发现, 很多网站的页面都是运用一定的页面模板实现的。在模版中会包含大量的该网站的索引超链, 而这些超链会跟随模版被继承到该网站的每一个网页中。显然, 这些超链不应该被考虑。

2) 有些大型网站(含有大量的网页)的主页会被本站点的其他网页大量链接, 而获得很高的 LHN, 尽管它有可能被极少的其他网站所链接。

3) 还要考虑网页编辑的欺骗行为。例如, 它们在某些网页中包含大量的不可见链接指向自己的页面, 进而获得较高的 LHN。

在下面的部分, 将总的 LHN 值称作原始 LHN, 将被其他站点链接的 LHN 值称为 LHN。

使用如上所述的策略, 新网页将会面临不公正的待遇, 因为一个新的网页, 即使质量很高, 由于知道它的网页编辑很少, 也只能得到很小的 LHN 值。它们需要时间, 才能被其他的网页编辑了解和评价。因此, 对于新的网页, 应用 LHN 去评价它们的优劣是不合适的。既然已经采用了 LHN 的方法, 就应该按照一定的算法对新网页给以 LHN 的补偿。以下就是 LHN 补偿算法。

如果使用 UNIX 系统的时间格式(自 1970 年以来的秒数), 可以获得网页的发布时间  $T(p)$ , 如果令当前时间为  $T_{\text{now}}$ , 补偿的阈值时间为  $T_u$ , 用如下公式获得补偿权值:

$$\text{WLT}(p) = \begin{cases} \lg(T_{\text{now}} - T_u) - \lg[T_{\text{now}} - T(p)] & \text{if } T(p) > T_u \\ 0 & \text{otherwise} \end{cases}$$

考虑了补偿权值后, 得到新的 LHN 值

$$\text{WL}'(p) = \lg[\text{LHN}(p) + 1] + \text{WLT}(p)$$

用  $\text{WL}_{\text{max}}$  来表示对于系统所有的  $p$  的  $\text{WL}'(p)$  的最大值, 采用如下公式将 LHN 值进

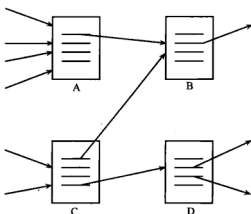


图 9-4 网页的互联结构示意图

归一化,得到期望的超链权值

$$WL(p) = \frac{WL'(p)}{WL_{\max}}$$

### 三、收集用户反馈信息

在搜索引擎中,当用户给出查询并得到一个返回结果列表之后,绝大多数的情况下他们都是扫描一下前面几个条目的摘要,感觉有他需要的内容,则点击对应的链接,去阅读网页全文。对来自于不同用户的同一个查询词来说,若某个链接虽然在返回结果表上出现的位置不太靠前,但被选取点击的次数比较多,于是系统应该感到该链接是比较受欢迎的,其位置应该往前调。举例来说,如果 80% 输入查询词“北京大学”的用户都点击了输出结果的第 10 项,则系统应该认为第 10 项对于查询“北京大学”来说才是最相关的结果,应该将它排在前面。

具体实现起来,我们的办法是通过用户点击数(User Hit Number, UHN)来达到这一目的。对于一个检索  $q$ ,会得到很多的检索结果网页,将这些网页表示为  $p_0, p_1, \dots, p_n$ 。假定检索  $q$  在一天内被提交了  $m$  次,定义  $WUH_i$  如下:

$$WUH_i(q, p) = \begin{cases} 1 & \text{if hit by } i\text{th User} \\ 0 & \text{otherwise} \end{cases}$$

之后,定义检索  $q$  对应的一个网页  $p$  的 UHN

$$WUD'(q, p) = \sum_{i=1}^m WUH_i(q, p)$$

但是,如果按照上述方法来统计用户评价权值的话,就忽略了返回结果中 URL 的位置信息。按照天网的统计,47.3% 的用户只访问天网返回给他们的第一页(包含 10 个结果),12.2% 的用户会继续访问第二页。这意味着一个结果在返回网页中的位置将会很大程度的影响用户点击的可能性。因此,如果一个结果网页在返回网页中排在尾部,即使这个网页和检索的相关度非常之高,它都只有很小的可能性被用户看到并点击。我们以一种补偿算法来弥补这个缺陷。这就是按照用户对每个返回页面访问的概率进行补偿。补偿因子  $c(\text{pos}(q, p))$  按照表 9-3 来定义。考虑到补偿因子,定义计算公式(9-1)。

表 9-3 补偿因子定义表

页面号	$\text{pos}(q, p)$	被用户点击的概率/%	$c(\text{pos}(q, p))$
1	1~10	47.3	1.00
2	11~20	12.2	3.88
3	21~30	7.4	6.39
4	31~40	5.0	9.46
5	41~50	3.7	12.78

续表

页 面 号	pos( $q, p$ )	被用户点击的概率/%	$c(\text{pos}(q, p))$
6	51~60	2.9	16.31
7	61~70	2.4	19.71
8	71~80	2.0	23.65
9	81~90	1.7	27.82
$\geq 10$	$\geq 91$	$\leq 1.4$	40.00

$$WUD(q, p) = \sum_{i=1}^n [c(\text{pos}(q, p)) \times WUD_i(q, p)] \quad (9-1)$$

公式(9-1)定义了检索  $q$  在一天的时间内,其结果页面  $p$  得到的用户评价,但是,这只考虑了一天的情况。应该如何来考虑长时期的用户评价呢?考虑  $n+1$  天的数据:  $WUD_0, WUD_1, \dots, WUD_n$ 。最简单的方法是将这些数据求和得到这段时间的用户评价,即

$$WUA'(q, p) = \sum_{i=0}^n WUD_i(q, p) \quad (9-2)$$

但这是不合理的,因为用户在不同的时间感兴趣的网页是不同的。例如,当用户在奥林匹克运动会召开之前检索“奥林匹克运动会”,他会大量的点击那些讲述有关奥林匹克运动会准备情况和参赛运动员情况的网页。在奥林匹克运动会召开的过程中,大量的用户更关注关于世界纪录被打破的情况,各个国家获得的奖牌数和排名情况。当奥林匹克运动会结束后,他们的兴趣会转向一些有关奥林匹克运动会的评论的网页。如果仍使用公式(9-2),当用户在奥林匹克运动会进行时检索,他们将会看到大量的讲述有关奥林匹克运动会准备的网页排在前面,因为这些网页曾被给予了很高的用户评价。为了避免这个问题,使用一种衰减算法

$$WUA(q, p) = \sum_{i=0}^n (k^i \times WUD_i(q, p))$$

这里的  $k$  是衰减系数。系数  $k$  的值越大,先前的数据对结果的影响就越大。 $k=0$  和  $k=1$  是两个极端情况。 $k=0$  表示历史的数据不被考虑; $k=1$  表示所有的历史数据都和现在的数据有相同的重要性。

搜索引擎面对的是数以亿计的用户,而上式需要搜索引擎保留所有的历史数据,这样的代价十分巨大。所以通过转化成公式(9-3)来解决这个问题。

$$\begin{aligned} WUA(q, p) &= \sum_{i=0}^n (k^i \times WUD_i(q, p)) \\ &= k^0 \times WUD_0(q, p) + k^1 \times WUD_1(q, p) + \dots + k^n \times WUD_n(q, p) \\ &= WUD_0(q, p) + k \times WUA_1(q, p) \end{aligned} \quad (9-3)$$

利用一个递归程序,只需要记录两个数据:历史数据  $WUA_1$  和当前数据  $WUD_0$ 。

当一个新的网页刚刚被索引时,它没有被用户点击的机会,所以如果采用上述的方法时,它的用户评价价值会是零,需要给予它们一些补偿。办法是给一个新的网页一个缺省的用户评价价值

$$WUC(q, p) = l \times WUA_{\max}(q)$$

补偿系数  $l$  反映了对于一个新的网页的重视程度。 $WUA_{\max}(q)$  是对于所有的  $p$  取值最大的  $WUA(q, p)$  值。考虑了补偿之后,新的用户评价价值由公式(9-4)计算:

$$WU'(q, p) = WUA(q, p) + WUC(q, p) \quad (9-4)$$

仍然需要对这个值进行归一化得到公式(9-5),其中  $WU_{\max}$  代表对于所有的  $p$  的  $WU'(q, p)$  的最大值。

$$WU(q, p) = WU'(q, p) / WU_{\max} \quad (9-5)$$

#### 四、计算最终的权重

以上已经给出了如何得到一个检索的相关网页集合,下面的工作是计算每个网页和查询  $q$  的相关度。相关度运算依赖三个方面:基本权值、链接权值和用户评价权值。首先计算每一个结果网页  $p$  的基本权值  $WB(q, p)$ 。

按照第一节的论述,每一个查询  $q$  可以被分解为  $m$  个特征项  $\{t_1, t_2, \dots, t_m\}$  的逻辑运算。因此,对于每一个结果网页  $p$ ,都可以获得每一个特征项在该网页中的权值  $WB(t_i, p)$ 。按照如下方法定义权值的逻辑运算:

$$\bigcap_{i=1}^m WB(t_i, p) = \sum_{i=1}^m WB(t_i, p) / m$$

$$\bigcup_{i=1}^m WB(t_i, p) = \max_{i=1}^m (WB(t_i, p))$$

任何一个用户的检索都可以表示为特征项的与( $\cap$ )和或( $\cup$ )的运算表达式,因此得到相应的权值运算公式(9-6)。

$$WB(q, p) = R(WB(t_0, p), WB(t_1, p), \dots, WB(t_m, p)) \quad (9-6)$$

$R$  就是相应的逻辑运算表达。根据公式(9-2),通过一定的运算过程,可以得到一个网页  $p$  的对应查询  $q$  的基本权值  $WB(q, p)$ 。

对于一个查询  $q$ ,还需要考虑链接权值  $WL(p)$  和用户评价权值  $WU(q, p)$ 。有两种方法来处理这三者之间的关系:

- 1) 让  $WB(q, p)$  作为基准权值,而链接权值和用户评价权值作为比例系数

$$W(q, p) = WB(q, p) \times WL(p) \times WU(q, p)$$

- 2) 每一种权值按照一定的比例重新构成新的权值

$$W(q, p) = k_{WB} \times WB(q, p) + k_{WL} \times WL(p) + k_{WU} \times WU(q, p)$$

其中

$$k_{WB} + k_{WL} + k_{WU} = 1$$

天网选择了第二种方法。在这种方法里,每种权值都起到了影响结果权值的作用

用,但是,它们的影响又都被限制到一定的范围内。第二种方法的优点如下。

1) 几乎所有的网页拥有者,尤其是商业网站,期望它们的网页被排在搜索结果的前列。一些网站就利用一些手段来欺骗搜索引擎。例如,它们通过向网页中加入一些不可见的文本来提高它们基本权值部分的值:一个旅馆为了扩大影响,将它们的主页中加入大量的不可见的词汇“计算机”。如果这样,当用户检索计算机时,这个旅馆的主页将堂而皇之的排在前几个结果中。利用第二种方法可以在一定程度上避免这种欺骗。

2) 如果忽略一个站点内部的链接,这就使得网站的作者很难通过超链权值对搜索引擎进行欺骗。一般来讲,其他网页的编辑不会愿意以牺牲自己网页质量的代价来将一些不相关的超链加入到自己的网页中。

3) 用户评价权值也是一个容易被用来欺骗搜索引擎的特性。一些网站的所有者或许会雇佣一些职员,不停地在检索结果页面中点击它们自己的网页。如果按照第一种方法,经过一定的时间,这个总被点击的网页最终将升到第一的位置。但如果使用第二种方法,就可以有效地避免这个问题,因为用户的影响已经被限定到了一个合理的范围。

总的来讲,结果排序是搜索引擎技术最重要的一个方面,从概念上讲,主体是本章第二节讨论的那些因素,但在实际系统中会衍生许多变化,其细节常常是商业机密。

## 第四节 信息检索技术评估

广义上的评估,在人们实际生活中有很多应用。例如,计算机性能评估是计算机系统结构课程中很重要的一部分内容;人们学习技术性的东西,都希望知道其到底怎么样,这涉及评估;在社会生活中,人们也常常需要评价人与事情的优劣,这同样用到评估。不同情况下,评估的内容和方法都会有所不同,在 IR(信息检索)领域,“评估”也有自己的特点。IR 是运用文本处理技术,形成面向普通的、大量的用户的系统。因此对 IR 系统进行评估,是用户对体现了某些技术特性的系统返回网页进行的一种评判,属于技术评估的范畴。由于文本语义的丰富性及文本集合本身的变化性、用户需求的多样性及系统模型本身的简单性,使得 IR 评估具有一定困难。但是,IR 系统评估确是必要的,它可以帮助我们区分技术的好坏,从而评判采用哪些技术形成的系统能够更好的适应用户的需求。

评估的目的是对用户负责,但是对系统的评估常常是基于对所包含各项技术评估基础上的,即无论是从研究方法还是工程实践的角度,都需要能够评估各个单项技术对系统行为可能的贡献。更进一步地,若能将设计指标和评估指标结合起来则最好。换个角度说,系统是由若干技术组成,设  $\text{System} = t_1 + t_2 + \dots + t_i$ ,某人发明了一个新的技术  $t_i$ ,如改进了一个算法,希望评价改进到底好还是不好,这里有两种办法:



一种是用它替换系统中的相应技术,看对总的效果的贡献;二是单独在一个评测环境中评估。前者涉及真正的系统,时间和费用的代价通常较高,并不是每个人都有这种条件的。例如,研究搜索引擎排序算法的人很多,但是接触真正的、有大规模用户的搜索引擎机会有限。后者是在一定的评测环境或者在一定规模的实验室中作评价,但实验的系统可能和真实系统应用有距离。本节讲述的是技术评估,但是最后要反映在系统总的效果上面。

一般而言,技术评估有如下三个层次。

1) 系统表现。评估中用户关心若干事情,记做  $F = \{f_1, f_2, \dots, f_n\}$ , 其中的  $f$  可以是相关性,新颖性,完整性,速度等,用户不关心向量模型,概率模型,关心的是应用层面的东西。

2) 测试指标。特别是在实验室环境中的测试,也会测试一些指标,记做  $G = \{g_1, g_2, \dots, g_m\}$ 。评估代价等原因使实现完整的评估比较困难,我们希望能用比较小的代价完成对它们的测试。用户关心的往往是定性的东西,一种比较直观的感觉,但是测试需要有一些比较客观的指标,而且希望对  $G$  的测试结果和  $f_1, f_2, \dots, f_n$  有很好的对应。

3) 设计指标。在设计系统的时候,用  $P = \{p_1, p_2, \dots, p_k\}$  表示实现程度对  $g_1, g_2, \dots, g_m$  贡献的关系,使得设计者心中有数。

系统设计指标和评估指标往往不一样举几个例子来说明:

1) 评估汽车。用户关心:速度,启动加速度,刹车距离等;发动机测试指标:转速,扭矩,马力等;发动机设计指标:排量,气缸数,点火方式等。汽车工业的实践已经证明了它们之间对应关系的有效性(尽管不是 100%)。

2) 评估计算机硬件。用户关心:速度、性能测试指标(SPEC benchmark)、设计指标(包括主频、CPI(Cycles Per Instruction)、字长和 Cache 大小)等。计算机工业的实践也基本证明它们的对应关系是有效的。

3) 评估大学。用户(社会)关心:培养能够为人类社会的进步充分发挥潜能的人;测试指标(包括政府高级官员的数量,大文豪、大科学家的数量等);设计的具体追求(包括得奖数、发表论文数(影响因子)、博士学位获得者人数和科研经费数等)。

上述例子说明,用同样的方法,在不同的情境下可以解释不同的问题,该方法的优劣还需要在实践中进行总结,在此不做更多评价。

不断地对现有的  $F, G, P$  及其相互关系提出疑问、提出改进,是研究评估的人们应该考虑的基本任务。我们需要了解现在的  $F, G, P$  是些什么(如何定义的,如何能得到),但仅此不够;事物都是在一个“目的”和“手段”链中发展的,这种链接关系的紧密程度就决定了达到目的的优化程度(效果、效率)。

信息获取技术评估模型如图 9-5 所示。其中,用户关心的指标有相关性、新颖性、完整性、速度等;信息获取的测试指标包括查全率、查准率、冗余度、 $F, E$  等;系统设计的追求指标包含布尔模型、 $TF * IDF$ 、概率模型、新词学习速度等。这三种指标

之间存在一定的联系,通常设计指标是比较微观的( $q, d$ ),测试指标是宏观的( $Q, D$ )。总体上,一般使用定性指标。测试和设计指标都是定量指标,图中问号表示的就是定量在定性指标上的反应方法。例如,查全率可能与完整性存在某种联系,而查准率可能与相关性有关联等。

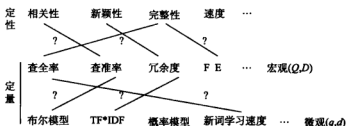


图 9-5 信息获取技术评估的“森林”

以下首先介绍信息检索技术评估指标;然后讲解信息检索技术评估实践,结合 TREC 评估和天网组制作的 CWT100g 测试集来讲解如何真正做一次大规模的信息检索技术评估;最后介绍针对搜索引擎技术的评估。

## 一、信息检索技术评估指标

在信息检索中有很多评估指标,应用最多的是准确率(Precision,  $P$ )和召回率(Recall,  $R$ ),其他的方法都是基于这两个指标引申而来。尽管这两个指标应用广泛,但还是有很多学者对它们提出了看法,因此需要用批判的眼光来看待这些方法。

### 1. 准确率和召回率

先来看准确率和召回率的定义和计算。要评价技术  $T$ ,对于给定的查询<sup>①</sup>, $q$ ,总体文档集合  $D$ 。记:

- 1)  $R$ :  $D$  中和  $q$  相关文档的集合,  $|R|$ :  $R$  的大小;
- 2)  $A$ : 算法  $T$  检索的文档集合,  $|A|$ :  $A$  的大小;
- 3)  $|Ra|$ :  $R$  和  $A$  交集的大小。

查准率和召回率基础定义如图 9-6 右侧所示。查准率定义为检索到的相关文档占有检索文档集的比例,或者说一个检索到的文档是与查询相关的概率;召回率定义为检索到的相关文档占有相关文档集的比例,或者说一个相关文档被检索到的概率(Saracevic 1995)。计算公式为图 9-6 左侧所示。

<sup>①</sup> 在 TREC 中通常称为主题(Topic),也即用户需求(user's need),查询是由主题变换而来的,可以采用人工和自动的方法。

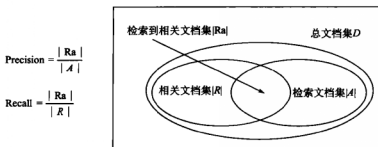


图 9-6 查准率和召回率基础定义图示

在此,需要考虑两个问题:①定义了,不等于能够计算。如何算得针对 $(Q, D)$ 的 $P, R$ ? ②人们认为在一定的排序意义上(ranked A)考察 $P, R$ 更有意义。沿着这个排序,考察“查准率随召回率变化的情况”。所以 $P, R$ 不是在原始意义上的,而是在排序上的计算。之所以这样做,是因为符合许多 IR 系统的实际情况,通常都会在结果集合上定义一个序。

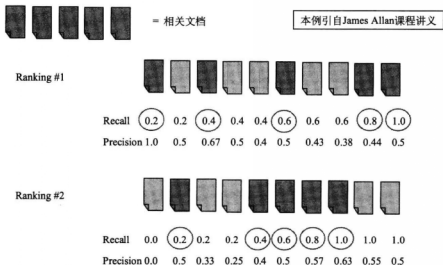


图 9-7 查准率和召回率例子

下面来看一个计算 $P$ 和 $R$ 值的例子,微观上的具体做法如图 9-7 所示。设相关文档数为 5 个,在 Ranking #1 和 Ranking #2 中,画圈标识的 $R$ 值所对应的上面文档表示相关,否则不相关。则按照 $R$ 和 $P$ 定义,可以计算得到各返回文档位置对应的 $R$ 和 $P$ 值。如 Ranking #1 中,第一个返回文档相关,则 $R$ 为 $1/5=0.2$ ;此时注意 $P$ 值应该考察“查准率随召回率变化的情况”,所以 $P=1/1=1.0$ 。以此类推,Ranking #1 中,第三个返回文档相关,则 $R=2/5=0.4$ ,此时 $P=2/3=0.67$ 。顺着 Rank-

ing 往下走,  $R$  和  $P$  有很多变化情况, 对应很多对数字, 将这些数字对应点画在坐标系中可以形成一条曲线。用的时候通常求算术平均值, 称为 MAP (Mean Average Precision), 就是对  $R$  增加时对应的  $P$  求平均, 单个查询  $q$  的平均准确率是每篇相关文档检索后的准确率的平均值。由此得到:

Ranking #1 的 MAP =  $(1 + 0.67 + 0.5 + 0.44 + 0.5) / 5 = 62.2\%$

Ranking #2 的 MAP =  $(0.5 + 0.4 + 0.5 + 0.57 + 0.63) / 5 = 52\%$

MAP 试图用一个数值概括  $R$  和  $P$  很多对应数字构成的曲线, 但是平均数有时候会掩盖一些矛盾, 如看起来 Ranking #1 的 62.2% 好, 如果用户只关心排序结果, 从图中可以看出 Ranking #2 返回了所有 5 个相关文档, Ranking #1 则只返回了 3 个。

为了使计算得到的值与人的直觉尽量符合, 即在一定意义上可用, 通常采用针对 11 点标准召回率的精度计算值 (eleven-point average precision)。人们建议在一些特殊的点上给出  $R$  和  $P$  的关系: 例如, 得到  $R=0\%, 10\%, \dots, 100\%$  时对应的  $P_r$ , 就能很方便地讲“召回率为 20% 的时候精度为  $X$ ”, 同时也可以使用“3 点标准”的说法: 25%, 50%, 75% 等。

按照这种想法, 如果  $D$  中相关文档的个数是 10 的倍数, 且如果算法给出的“Ranked A”包含了所有相关文档, 则会容易得到这些点; 否则要考虑如何插值的问题。

先来看一个“省事的”11 点标准召回率例子, 如图 9-8 所示。

1)  $D = \{d_1, \dots, d_{1000}\}$ , 对查询  $q$ , 所有相关文档集合共 10 个元素,  $R_q = \{d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123}\}$ 。

2) 查询的返回结果序:  $d_{123}^*, d_{84}, d_{56}^*, d_6, d_8, d_9^*, d_{511}, d_5^*, d_{39}^*, d_{129}, d_{187}, d_{25}^*, d_{38}, d_{44}^*, d_{57}, d_{71}^*, d_{48}, d_{250}, d_{113}, d_3^*, d_{200}, d_{144}, d_{11}, d_{89}^*, d_1$ 。

3) Ranking: \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \* ~ \*

Recall: .1. 1. 2. 2. 3. 3. 4. 5. 5. 5. 6. 6. 7. 7. 8. 8. 8. 8. 9. 9. 9. 9. 1 1。

Precision: 1. 5. 67. 5. 4. 5. 43. 5. 55. 5. 45. 5. 46. 5. 46. 5. 47. 5. 42. 45. 43. 41. 39. 42. 4。

但实际操作中经常出现这样的问题。一是得到的结果集合不包含所有的相关元素。实践上常常只是返回排序较高的若干元素, 因此不能得到需要的  $R$  值。二是  $D$  中相关元素的个数不是 10 的倍数。于是能直接得到的  $R$  值不一定包含 0%, 10%, 20%, 30%, ..., 100%。

再来看一个实践中召回率例子, 如图 9-9 所示, 后面没有出来的相关文档

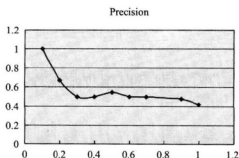


图 9-8 “省事的”11 点标准召回率例子

$P$  值按算零, 零不是算法本身的特点, 可能是考虑性能, 节省资源等原因, 只返回一定数量的文档。故为了公平起见给出了两种图示。考虑到:

- 1) 所有相关文档集合 (共 10 个元素):  $R_q = \{d_3, d_5, d_9, d_{25}, d_{39}, d_{44}, d_{56}, d_{71}, d_{89}, d_{123}\}$ 。
- 2) 只能得到 5 个有效的  $R$  值: 10%, 20%, 30%, 40%, 50%。
- 3) 对查询  $q$  返回的结果序:  $d_{123}^*, d_{34}, d_{56}^*, d_6, d_8, d_9^*, d_{511}, d_{129}, d_{187}, d_{25}^*, d_{38}, d_{48}, d_{250}, d_{113}, d_3^*$

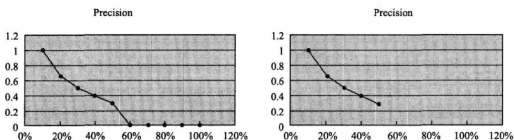


图 9-9 实践中召回率例子

鉴于上述情况, 需要插值(interpolation), 目标是在 11 个标准点  $R$  上都有  $P$ 。

1) 可以想出各种“合理的”方法, 如将已知的点连起来, 不同的方法结果会不一样, 因此做比较时要讲清楚。下面是经常推荐的两种方法。

•  $P(r_j) = \max P(r), r_j \leq r \leq r_{j+1}$ 。取在下一个标准召回率之间的已知召回率对应的最大精度值, 不一定保证单调性。

•  $P(r_j) = \max P(r), r_j \leq r$ 。取往后的已知召回率对应的最大的精度值, 这样得到的是阶梯函数, 保证单调性。

2) 如果需要, 还要考虑返回的结果不包括所有相关文档的情况, 如令其后的标准点上的  $P=0$ 。

前面讲到的是针对单个  $q$ , 我们最终关心的是对  $Q$  的总体情况评价, 即查询集合的 MAP (每个  $q$  的平均准确率的平均值), 计算公式为

$$\bar{P}(r_i) = \sum_{i=1}^{N_q} \frac{P(r_i)}{N_q}$$

其中,  $r_i$  取标准召回率,  $N_q$  是所考察  $Q$  的大小。这样就得到一个技术 (算法) 在  $(Q, D)$  上精度的宏观表现。

## 2. 调和平均值 $F$

用  $P, R$  11 点标准图反映一个查询集合上的平均性能的方法, 在评价一个 IR 算法性能好坏中有时不尽如人意, 因此尝试用一个数值来表示  $P$  和  $R$  的综合效果, 即

调和平均值  $F$ , 计算公式如下。

$$\text{调和平均 } F \text{ 指数: } F = H(P, R) = \frac{2}{\frac{1}{P} + \frac{1}{R}} = \frac{2PR}{P+R}$$

那么, 为什么不采用算术平均? 即  $F = A(P, R) = \frac{P+R}{2}$

这是考虑到  $A(P, R)$  和  $H(P, R)$  并不一致(当然也可以举出它们一致的例子), 例如:

- 1)  $P_1=0.1, R_1=0.83; A(P_1, R_1)=0.42, H(P_1, R_1)=0.197;$
- 2)  $P_2=0.3, R_2=0.3; A(P_2, R_2)=0.3, H(P_2, R_2)=0.3;$
- 3) 也就是说,  $A(P_1, R_1) > A(P_2, R_2)$ , 但  $H(P_1, R_1) < H(P_2, R_2)$ 。

具体地, 如果系统检索到了全部文档集, 则所有相关文档都会在其中, 此时  $R=1$ ; 但是通常情况下全部文档集中只有一小部分文档是与查询相关的, 此时  $P$  值会非常小, 接近 0。所以评测指标定义的倾向, 在一定情况下, 希望同时获得较高的  $P$  值和  $R$  值。分子是乘积, 如果要最大化, 那么  $P$  和  $R$  相等的时候最大。所以说, 调和平均指标不会掩盖  $P, R$  一个方面特别的不足(如上例中的  $P=0.1$  就很小), 如果用算术平均就把它掩盖了, 因此采用调和平均值  $F$ 。

上式是针对一个  $q$ , 而我们最终关心的是对  $Q$  的总体情况计算  $F$ 。故采用如下两步完成。

- 1) 对每一个查询  $q$ , 得到标准召回点上的  $F$ , 即

$$F_q(i) = 2 * p[i] * r(i) / (p(i) + r(i)), i = 0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1$$

在查询内求平均(micro-average):  $F_q = \sum F_q(i), i = 0, .1, .2, \dots, .9, 1$ 。

- 2) 进一步在查询间求平均(macro-average):  $F = \sum F_q, q \in Q$ 。

在实际情况中, 有些用户追求  $P$ , 如希望搜索引擎返回结果的第一个页面中就包含相关文档; 有些用户追求  $R$ , 如希望检索到尽可能多的相关文档。鉴于此, van Rijsbergen(Rijsbergen 1979)给出了  $E$ (Effectiveness)指标, 它允许使用者指定  $P$  和  $R$  的相对重用性:

$$E \approx 1 - \frac{1}{\alpha(\frac{1}{p}) + (1-\alpha)\frac{1}{R}}$$

$\alpha$  参数取值从 0(用户不关心  $P$ )到 1(用户不关心  $R$ )。当  $\alpha$  取 0.5 的时候, 是用户认为  $P$  和  $R$  同等重要。

### 3. 针对 $P, R, F$ 总结

下面讨论, 对于如下给定的条件和变量: ①包含一个新算法的 IR 系统(测试); ②一个测试文档集合  $D$ ; ③一个查询集合  $Q = \{q\}$ ; ④一个事先确定的相关集合的集

合  $G(Q)$  (即对每一个  $q$  都要有相关集合, 如何确定这个算法的  $P-R$  图和  $F$  值呢?  
计算流程如下。

1) 对于  $Q$  的每一个元素  $q$ :

- 得到一个有序结果集  $s(q) = \langle d_1, d_2, \dots, d_q \rangle$ 。
- 与  $G(q)$  对比, 依序计算  $s(q)$  中元素的  $r[i]$  和  $p[i]$ ,  $i=1, 2, \dots, q$ 。
- 选择一种合适的插值方式, 得到  $p[i]$  在  $r=0, .1, .2, .3, .4, .5, .6, .7, .8, .9, 1$  处的插值。如果  $r[q] < 1$ , 则令它之后的标准点上的  $p=0$ 。

2) 对  $Q$  的所有元素, 在标准召回点上求  $p$  的平均值。

3) 给出平均值的统计表和  $P-R$  图。如图 9-10 所示, 左侧是统计表, 右侧是  $P-R$  图。这是实际中的 44 个查询, 有的是词, 有的是短语, 其中  $P-R$  图是递减的。因为在 IR 系统中返回结果的序列, 如果用 1 表示相关, 0 表示不相关, 出现 1 的概率要比后面的高。

Precision - 44 queries		
Recall	Terms	Phrases
0	88.2	90.8
10	82.4	86.1
20	77.0	79.8
30	71.1	75.6
40	65.1	68.7
50	60.3	64.1
60	53.3	55.6
70	44.0	47.3
80	37.2	39.0
90	23.1	26.6
100	12.7	14.2
Average	55.9	58.9

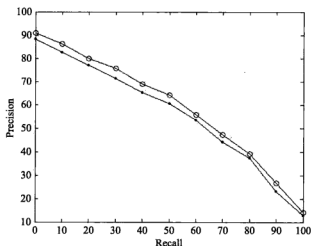


图 9-10 实际中的 44 个查询词的评价统计表和  $P-R$  图

即使会计算  $P, R, F$  值, 还有很多问题值得继续探讨和研究, 如这些指标同现在搜索引擎相关性到底有多大? 什么样的指标体系对搜索引擎是最好的?

此外, 目前面临困难包括:

- 1) 有可能  $D$  和  $Q$  太大, 得出  $G(Q)$  代价太高。
- 2) “相关”的含义因人而异。
- 3) 如此定义的  $P, R, F$  适于“批处理”评估, 没有体现现代 IR 系统的典型特征

——交互式信息检索过程。

4) 如此定义的  $P, R, F$  依赖于返回结果的线性序, 但有些系统不一定有这样的序。

5) 如果对象是搜索引擎, 什么是评估搜索引擎排序算法最好的方法呢?

#### 4. R-Precision, $P@N$ , MRR 和 Bpref 指标

针对不同的评测目的, 存在从  $P$  和  $R$  衍生出来的其他评测指标。

##### (1) R-Precision

单个查询的 R-Precision 是检索出  $R$  篇文档时的准确率。其中,  $R$  是文档集中与查询相关的文档的数目。查询集合的 R-Precision 是每个查询的 R-Precision 的平均值。例如: 假设有两个查询, 第 1 个查询有 50 个相关网页, 第 2 个查询有 10 个相关网页, 某个系统对于第 1 个查询返回的前 50 个结果中有 17 个是相关的, 对于第 2 个查询返回的前 10 个结果中有 7 个是相关的, 则该系统在第 1 个查询上的 R-Precision 为  $17/50=0.34$ , 在第 2 个查询上的 R-Precision 为  $7/10=0.7$ , 查询集合的 R-Precision 为  $(17/50+7/10)/2=0.52$ 。

##### (2) $P@N$

单个查询的  $P@N$  ( $N$  可以等于 10, 20, 或者 30, ...) 是系统对于该主题返回的前  $N$  个结果的准确率。查询集合的  $P@N$  是所有查询  $P@N$  的平均值。通常  $N$  取 10。

##### (3) MRR

单个查询的  $RR$  (Reciprocal Rank) 值是第一个正确答案出现位置的倒数, 如果没有检索到则为 0。例如, 系统返回的第一正确答案排在 4, 则  $RR=0.25$ 。查询集合的  $MRR$  (Mean Reciprocal Rank) 是所有查询  $RR$  的平均值。

$MRR$  指标用在评测 Web 检索任务中的 HPNP 搜索 (Home Page/Named Page 主页和指定页面查询/导航搜索)。HPNP 搜索是指用户用名称来查询特定的页面, 通常正确答案只有一个。在实际中, 由于 URL 别名和网页复制, 可能有多个答案, 此时  $RR$  取排序最靠前的文档位置的倒数。

##### (4) Bpref

各种评估方法不是等可靠的。有的评估方法固有就比其他的方法稳定, 如 MAP 错误率比  $P@10$  的低一半。当相关结果集不完整的时候, 原有的依赖完整结果集的方法不强壮。

实验的错误率是指在重复实验中, 不同次实验得到相反的评估结果的次数与实验总次数的比率, 它代表一次评估实验得到错误结果的可能性。(Buckley et al. 2000) 给出如下计算公式:

$$\text{ErrorRate} = \frac{\sum \min(|A > B|, |A < B|)}{\sum (|A > B| + |A = B| + |A < B|)}$$



其中  $A, B$  为评估对象,  $|A > B|$  表示重复实验得到  $A$  优于  $B$  的次数。如果实验在某一变化因素控制下, 计算错误率, 可以用来评估实验对这一变化因素的稳定性, 如对评测员、查询集合大小以及评测指标等。

$P@10$  只考虑前 10 个检索文档中的相关文档, 考虑到用户希望返回的第一页结果页面(通常 10~20 条)中包含尽可能多的相关网页, 容易理解这个评估方法与用户使用搜索引擎的满意度关联紧密。但是不同的查询对应的相关文档数目是不一样的, 采用常量 10 作为所有查询的截断是不合适的。R-Precision 解决了  $P@10$  采用常量截断的问题。MAP 利用了比 R-Precision 和  $P@10$  更多的信息。因此 Buckley 等(Buckley et al. 2000)得出结论:

1)  $P@10$  不是区分检索算法的有效指标, 与 P-Precision 和 MAP 相比它的误差幅度更大。

2) R-Precision 误差幅度小于  $P@10$ , 但是大于 MAP。

3) MAP 稳定性上更好一些, 即误差幅度三者中最小。它的缺点是不如  $P@10$  和 R-Precision 容易直观解释。

R-Precision,  $P@N$ , MRR 这三种方法都是基于文本集中的所有相关文档都已经找到, 即结果集完整, 但是当文本集增大无法获得所有相关文档的情况下, 它们的稳定性都急剧下降。评测方法的稳定性是指采用一个评测指标的时候, 在不同的测试集中检测所有参加评测的 IR 系统, 得到的所有相对评测结果序基本稳定, 没有大步距的调整。为了度量稳定性, 需要计算结果序的相关系数  $r$  来确定, 相关系数的取值在 -1 到 1 之间。相关系数取 1 表示保持一致, -1 表示不一致, 0 表示无关系。一般从经验上来说, 当时  $|r| \geq 0.8$  时可视为高度相关;  $0.8 > |r| \geq 0.5$  时可视为中度相关;  $0.5 > |r| \geq 0.3$  时可视为低度相关;  $|r| < 0.3$  时可视为不相关。

通常采用 Kendell  $\tau$  相关系数<sup>①</sup>来计算:

$$\tau = \frac{2P}{2n(n-1)} - 1$$

其中,  $n$  是一个排序中的元素个数,  $P$  是对两个排序中任意一个元素后续位置个数求和。

例如, 对一组 8 个人分别按照身高和体重排序, 考察身高和体重之间的相关性。

人	A	B	C	D	E	F	G	H
身高序	1	2	3	4	5	6	7	8
体重序	3	4	1	2	5	7	8	6

则对于体重序而言, 一个元素是 3, 在它的右侧有 5 个更高序的元素, 这个元素

<sup>①</sup> Kendell  $\tau$  的解释 [http://en.wikipedia.org/wiki/Kendall%27s\\_tau](http://en.wikipedia.org/wiki/Kendall%27s_tau)。

对  $P$  的贡献就是 5; 第二元素是 4, 它右侧有 4 个更高序的元素, 这个元素对  $P$  的贡献就是 4; 以此类推, 得到  $P = 5 + 4 + 5 + 4 + 3 + 1 + 0 + 0 = 22$ , 则  $\tau = 44/28 - 1 = 0.57$ 。

为了解决相关结果集不完整的时候, 原有的依赖完整结果集的方法不强壮的情况, 研究者们提出了 Bpref (preference-based, 基于优先选择) 方法, 不同于以前的标准, 它并不直接考虑返回的正确结果的精度和召回率, 而是看标准答案 (包括相关及不相关的文档), 在返回结果中相对位置的关系 (是否相关的排在了不相关的之前) 来排除未对其人工评价的文档造成的影响。计算方法如下 (Buckley et al. 2004):

$$\text{Bpref} = \frac{1}{R} \sum_r 1 - \frac{|n \text{ ranked higher than } r|}{R}$$

其中,  $R$  是对于一个查询地相关文档个数,  $n$  是当  $r$  个相关文档检索到时的不相关文档数。  $n$  对应的每一个文档属于 top  $R$  个判断为不相关的文档。 Bpref 对于相关文档过少 (如 1~2 个) 的时候过于粗糙, 改进的方法是 Bpref-10, 可以保证至少有 10 个判断过的文档:

$$\text{Bpref-10} = \frac{1}{R} \sum_r 1 - \frac{|n \text{ ranked higher than } r|}{10 + R}$$

Bpref 方法的实现可以参考网站 [http://trec.nist.gov/trec\\_eval/trec\\_eval\\_latest.tar.gz](http://trec.nist.gov/trec_eval/trec_eval_latest.tar.gz) 中 Bpref\_bug 文件, 其中有一段伪码。需要注意: 人工判断出来的答案集除了记录相关文档的列表, 还要记录被判断为不相关文档的列表。

Buckley 等 (Buckley et al. 2004) 证明 Bpref 方法在结果集完整的情况下与现有的方法高度相关, 在结果集不完整的情况下比现有评测方法稳定性高。

## 5. DCG 指标

DCG (Discounted Cumulative Gain, 递减权值累加) 是 Jarvelin 等 (Jarvelin et al. 2000) 提出的。其主要优势是:

1) 采用了多级相关性评估 (multiple degree relevance assessments)。因为高度相关的文档比部分或者更少相关的文档更有价值。前述评估方法都采用二元方式判断, 即使判断答案集中包含多级相关, 这些指标也把它们归为相关, 或是不相关。在实际中, 区分文档相关的“程度”是需要的, 即把文档相关性引入评估方法对于用户关心的高度相关文档才是公平的。

2) 考虑文档在检索结果中的位置。相关文档的位置越靠后, 用户看到的机会越少, 从而价值越低, 因此需要考察文档在检索结果中位置的影响。

3) 与  $P$ - $R$  图比较, 更加直观并容易解释。

对检索到的文档赋予不同的权值, 同时根据文档序递减排权值, DCG 的计算方法:

$$DCG[i] = \begin{cases} G[1], & \text{if } i = 1 \\ DCG[i-1] + G[i]/\log_i i, & \text{otherwise} \end{cases}$$

其中  $b$  代表用户浏览结果的耐心程度, 可以取 2,  $e$ , 或者 10,  $b=2$  表示没有耐心的用户, 只浏览很少的结果;  $b=10$  表示有耐心的用户。通常  $b=e$ 。

例如, 把相关性分四个级别 (0, 1, 2, 3), 0 表示不相关, 1 表示沾边, 2 表示部分相关, 3 表示高度相关。把一个检索结果对应文档编号替换为相关性权值, 如

$$G' = \langle 3, 2, 3, 0, 0, 1, 2, 3, 0, \dots \rangle$$

当  $b=e$  时,

$$DCG' = \langle 3, 5, 6.89, 6.89, 6.89, 7.28, 7.99, 8.66, 9.61, 9.61, \dots \rangle$$

## 二、TREC 和 CWIRF 信息检索评估

有评测才会有鉴别。评判一种方法优劣的唯一标准是相互可比的评测, 而不是设计人员自己设计的自评, 更不是人们的直觉或某个人的意见。在信息检索的领域中, 检索系统评估对于系统的研究、设计与发展一直起着重要的作用。早期对检索系统评估最著名的研究是 Cleverdon 在 1950 年代末期开始进行的 Cranfield 测试 (Cleverdon 1997), 它开创了以测试集 (Test Collection) 配合测量准则 (Measures) 来评估系统的模式。所谓测试集, 是一种在规范化环境中测试系统效能的机制, 包括测试文档集 (Document Set)、查询问题 (Queries) 及相关判断结果 (Relevance Judgment) 三个部分。其研究设计的概念是基于在给定的查询问句与文件集中, 某些文件是与查询问句相关的假设。系统的目的是检索出相关的文件, 并拒绝不相关的文件。Cranfield 研究首开规范化系统评估之先河, 其评估模式也成为后世普遍采用的模型。然而, 正如前面所述, Cranfield 测试亦是在实验室环境下进行, 与实际的检索环境相差很多, 因此评估结果的实用性仍然受到许多质疑。为了促进信息检索的研究与应用的发展, 美国国家标准与技术局 (National Institute of Standards and Technology, NIST) 与美国国防部高等研究计划局 (Defense Advanced Research Projects Agency, DARPA) 共同举办了文件检索会议 (Text REtrieval Conference, TREC, <http://trec.nist.gov>), 通过提供大规模的大型测试集, 制定各种测试项目、提供统一的计分方法和评测软件, 组合成一个评估检索系统的机制, 以保证每个研究小组都能在一种公平、公开的条件下进行研究方法的探讨, 推动科学技术的进步。第一届 TREC 在 1992 年举办, 其后持续在每年年底举办会议, 至今 (2012 年) 已进行了 20 届, 正在举办第 21 届。除了与会者依据大会提供的测试集递交各测试项目的数据以进行评比之外, 还有为期数天的研讨会, 与会者可以在会中发表系统的架构、评估结果, 并相互讨论切磋 (TREC 2006)。

在信息检索领域中, 检索系统评估对于系统的研究、开发和应用一直有其显著的影响。目的是达到 TREC 追求的 4 个主要目标。

- 1) 以大规模测试集为基础,推动信息检索的研究。
- 2) 经由开放式的论坛,使与会者能交流研究的成果与心得,以增进学术界、产业界与政府的交流互通。
- 3) 经由对真实检索环境的模拟与重要改进,加速将实验室研究技术转化为商业产品。
- 4) 发展适当且具应用性的评估技术,供各界遵循采用,包括开发更适用于现有系统的新的评估技术。

Web 的发展伴随着信息的急剧增长,信息检索技术成为研究的热点,Google 在短短的几年里在全世界范围的成功,进一步印证了 Web 搜索是信息检索中的一个重要研究和应用方向。TREC 从 1992 年开始致力于信息检索评估工作,极大地推动了信息检索技术的发展。目前,国际上除 TREC 外,从事信息检索评估的还有 CLEF (Cross-Language Evaluation Forum, <http://www.clef-initiative.eu>), NTCIR (NII Test Collection for Information Retrieval, <http://research.nii.ac.jp/ntcir/index-en.html>), CWIRF (Chinese Web Information Retrieval Forum, 中文 Web 信息检索论坛, <http://www.cwirf.org>) 和 FIRE (Forum for Information Retrieval, <http://www.isical.ac.in/~clia>)。CLEF 是由欧洲各国的学者专家合作建构的评估机制; NTCIR 是日本情报学研究所 (National Institute of Informatics) 主办; CWIRF 是北京大学网络实验室主办; FIRE 是由印度统计学院 (Indian Statistical Institute) 主办。

TREC 是个一次持续一整年的信息检索评估活动,包括筹备工作、发布评估任务、参与者递交检索结果、评估人员进行文档相关性判断、送回评估结果和举办年度讨论会议等程序。关于 TREC 的历史以及演变在书 (Voorhees et al. 2005) 和文章 (江玉婷等 1998) 中有详细的介绍, NTCIR 和 CLEF 在文章 (陈光华 2004) 中有介绍。因为我们一直在参加 CWIRF 工作,所以下面主要结合 CWIRF 来介绍相关工作。

检索评估的基础是测试集, TREC 从 1992 年开始, 致力于信息检索评估工作, 提供英文 Web 测试集, NTCIR 提供日文 Web 测试集, CWIRF 提供中文 Web 测试集。

测试集在整个评估活动中所扮演的角色如图 9-11 所示, 从图中可以看出测试集、检索系统、评估人员与评估准则之间的互动关系。其中相关结果集的形成通过专门的评估人员人工判断, 在大规模数据集上进行检索系统评估时, 这种人工的工作成为最大的开销, 并使得对整个数据集进行完全的相关性判别成为不可能。Sparck Jones 和 van Rijsbergen (Rijsbergen 1975) 提出一种称为 Pooling 的方法构造, 它的假设是与查询相关的文档应该会被多数的检索系统检索出来。对每个评估用的查询, 从所有参加评估的结果序列中挑选出前一部分, 人工判断其相关性, 其他文档作不相关处理。这种做法的一个局限是, 当参加队较少的时候, 可汇集的结果权威性较差。提出 Pooling Plus 方法, 即将搜索引擎转换为虚拟参赛队, 参与结果集成。这

样,即使参赛队数量不多,也能合成质量较高的结果集,达到检验参赛系统检索质量的目的。

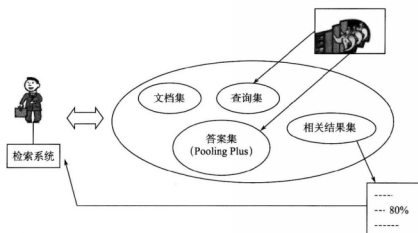


图 9-11 测试集在检索评估中的角色

北京大学计算机系网络实验室在 2004 年首次公布 CWT100g 中文 Web 测试集,包括测试文档集、查询问题及相关判断结果三个部分。它的时间跨度 9 个月,构建过程如下。

时间推进	2004. 2	2004. 6	2004. 10	2004. 11	2004. 11
任务	CWT100g 设计	构建文档集	构建查询集	Pooling	相关结果集

### 1. 构建文档集

根据天网搜索引擎截止 2004 年 2 月 1 日发现的中国范围内提供 Web 服务的 1 000 614 个主机,从中采样 17 683 个站点,在 2004 年 6 月搜集获得 5 712 710 个网页,包括网页内容和 Web 服务器返回的信息,容量为 90GB。其中每个网页对应的服务器返回信息中的 MIME 类型都是“text/html”或者“text/plain”。文档集数据格式参见(TianwangStorageFormat 2003)。

CWT100g 从 2004 年 6 月 16 日发布到 11 月份开讨论会,有超过 20 家单位申请此数据,包括中科院 3 家单位,北京大学信息中心,哈尔滨工业大学 2 家单位,保定明辉加盟公司,福建师大附中,北京大学语言所,北京大学计算研究所,清华同方光盘股份有限公司,上海交通大学,延边大学,TRS 公司,华南理工大学 3 家单位,江西师范大学,苏州大学计算机系,大连理工大学,中国农业科学院等。

## 2. 构建查询集

查询集是从 2002 年 4~6 月天网日志中采样获得待选集合,然后人工挑选编辑完成。针对中文 Web 信息检索评测的两个任务:主题提取(Topic Distillation, TD)和导航搜索(HP/NP,包括主页和指定页面查询),郭化楠等(郭化楠等 2004)的文章讲解了如何制作 CWT100g 的查询主题。3 个人在理解题目,达成一致的条件下,利用两周时间,总计形成 70 个 TD 的 topic,记在 TD1-70 文件中,285 个 HP/NP 的 topic,记在 HP/NP 1-285 文件中。其中主要利用的工具包括:①sohu 的分类目录,②天网检索技术搭建的 CWT100g 全文检索系统,③辅助工具(如利用 URL 特征和 anchor-text,找 HP/NP,找的时候同时记录答案)。

下面讲解中文 Web 信息检索评测的两个任务:TD 和 HP/NP。

### (1) 主题提取

主题提取的目的是对于一个特定主题发现一组关键资源。在任务中,将注重以站点作为资源的查询。要求是在前 10 个结果中寻找尽可能多的不同站点(用它们的网站的入口页面表示)。

例如,对于主题“linux”,CWT100g 中的下面站点可能被认为是关键资源:

<a href="http://www.oldlinux.org">http://www.oldlinux.org</a>	linux org
<a href="http://www.mhdn.net/os/29">http://www.mhdn.net/os/29</a>	明辉开发者网络 linux 区
<a href="http://www.redflag-linux.com">http://www.redflag-linux.com</a>	红旗 linux

关键资源页面原则上应该是一个“好的首页面”,那么如何判断一个站点的首页面是否是“好的首页面”呢?可以考察返回的页面是否符合如下三个标准:

- 是否大部分切合主题;
- 提供主题的可靠信息;
- 不是一个更大的切合主题站点的一部分。

例如,对于“linux”这一主题,页面“www.mhdn.net”不符合第一个条件,而页面“<http://www.redflag-linux.com/chanpin/Desktop/index.html>”不符合第三个条件。实例为:

```
<top>
<num>Number, TD3
<title>户外运动</title>
<desc>Description:
介绍户外运动的网站,比如装备、注意事项、团体。
</top>
```

### (2) 主页和指定页面查询

用户有时候会用名称来查询特定的网页。在这种情况下,一个有效的搜索系统将在第一个或前几个返回结果中给出相应的网页。

主页查询和指定页面查询,这两种情况下,查询结果只有一个并且用户的查询需求常常是页面的名称。不同的是主页查询的目标是一个特定的主页,而指定页面查询所找到的可以不是一个主页,而是满足用户需求的特定页面。例如,主页查询时,查询词“内蒙古民族大学”对应的查询结果是“www.mzdx.com”;而指定页面查询时,查询词“2001年中国十大并购人物”对应的查询结果是“www.mergers-china.com/top10/index-7.asp”。一些查询/排序策略对两种类型的查询都有效,而有的只对其中一种有效。

我们提供混和的查询集合。评判为:判断参与者返回文档的 URL 是否为最初选定文档的 URL。当然,如果页面有两个不同的 URL,那么这两个都将被认为是正确的答案。通过第一个正确答案的位置对系统进行比较。以第一个正确答案出现位置的倒数平均值(MRR)为评估准则。

### 3. 构建相关结果集

结合利用天网索引系统构建的 CWT100g 检索系统和 Pooling 方法构建结果集。各参加单位首先申请获得 CWT100g 数据,用各自的技术在 CWT100g 上建立一个查询系统,然后由评测组提供查询集给参加单位,各参加单位检索结果递交给评估组,形成 Pooling(包括 Google, Yisou, Baidu, Sogou, Zhongsou 五个搜索引擎的检索结果,与 CWT100g 取交集,形成虚拟参加评测队),人工结合计算机辅助的程序完成相关文档的判断。

在 SEWM-2004 中文 Web 信息检索评测中,报名截止日期 2004 年 10 月 20 日。10 月 21 日发查询测试集,10 月 31 日各参加队提交查询结果。共有 12 个队报名参加,7 个参加队提交结果(不包括北大天网,但给出结果作为参考,并现场演示)。各队提交结果如表 9-4 所示,第一列是参加单位名称,第二列是各队提交结果 RUN 的 ID,第三列是针对 TD 任务各队提交 RUN 的数量,第四列是针对 NPHP 任务各队提交 RUN 的数量。

表 9-4 2004 中文 Web 信息检索评测提交结果

TEAM	NAME	TD-RUNS	NPHP-RUNS
上海交通大学 APEX 实验室	APEX	5	5
北京大学计算机科学技术研究所	ANS	3	2
TRS 公司	TRS	5	2
华南理工大学木棉一队	MUMIAN1	3	1
华南理工大学木棉二队	MUMIAN2	2	1
华南理工大学计算机学院数据库应用研究室	SCUTDB	5	5
福建师大附中	WLL	0	1



## 相关度评测

CWT100G是SEWM2004的中文Web检索竞赛指定测试集。(→检索结果提交情况)

本项工作是对**主题提取任务**的检索结果进行相关性判别，生成CWT100G的相关结果集。首先请仔细阅读下面的判别准则：(节选自SEWM-2004中文Web检索测试指南)

主题提取目的是对于一个特定主题发现一组关键资源。在今年的任务中我们将只注重以站点作为资源的查询。要求是在前十个结果中寻找尽可能多的不同站点(用它们的网站的入口页面表示)。

例如对于主题“linux”，CWT100G中的下面站点可能被认为是关键资源：

http://www.oldlinux.org/ linux.org  
http://www.mhcn.net/os/29/ 明辉开发者网络 linux区

http://www.redflag-linux.com/ 红旗Linux

被判断为是一个关键资源，返回页面应该是一个站点的好的首页面。判断是否一个好的首页面应该考查结果是否符合下面三个条件：

- 1) 是否大部分切合主题；
- 2) 提供主题的可靠的信息；
- 3) 不是一个更大的切合主题站点的一部分。

对于“linux”这一主题，页面“www.mhcn.net”不符合第一个条件，而页面“http://www.redflag-linux.com/chanpin/Desktop/index.html”不符合第三个条件。

用户名:

图 9-12 帮助判断相关结果页面的计算机辅助程序入口



图 9-13 帮助判断相关结果页面的计算机辅助程序操作界面



针对 70 个 topic, 利用 Pooling 方法, 形成 35 000(70 \* 500) 个候选网页待检查。如果每人检查 4 个 topic 对应的网页, 工作量是检查 2000(4 \* 500) 个网页。大约需要 18(70/4) 个评估人员。

因为评估人员少, 没有按照 TREC 中每个 topic 至少 2 人原则来进行, 而是在统一做完后请 3 个专业评估人员检查。为了减少人工判断相关结果集, 使用了计算机辅助(说明参见图 9-12 和 9-13), 评测结果如下。表 9-5 是主题提取, 表 9-6 是导航搜索。

表 9-5 主题提取

R-Precision	P@10	RUN	R-Precision	P@10	RUN
0.206384	0.3200	MUMIAN1_RUN_2	0.084371	0.1300	APEX_RUN_3
0.214622	0.3040	MUMIAN1_RUN_1	0.077692	0.1220	APEX_RUN_1
0.211303	0.2760	MUMIAN1_RUN_3	0.106665	0.1140	TRS_RUN_2
0.165782	0.2320	ANS_RUN_1	0.101380	0.1140	TRS_RUN_3
0.157969	0.2300	ANS_RUN_2	0.100046	0.1140	TRS_RUN_1
0.157552	0.2060	ANS_RUN_3	0.096551	0.1140	TRS_RUN_5
0.127645	0.1800	MUMIAN2_RUN_2	0.061310	0.0800	TIANWANG_RUN_1
0.142538	0.1680	MUMIAN2_RUN_1	0.049262	0.0720	SCUTDB_RUN_1
0.101664	0.1420	TIANWANG_RUN_3	0.065224	0.0660	TIANWANG_RUN_2
0.090712	0.1400	APEX_RUN_5	0.044195	0.0540	SCUTDB_RUN_2
0.086950	0.1380	APEX_RUN_4	0.014442	0.0160	SCUTDB_RUN_5
0.087927	0.1340	APEX_RUN_2	0.011289	0.0120	SCUTDB_RUN_4
0.109163	0.1320	TRS_RUN_4	0.005764	0.0100	SCUTDB_RUN_3

表 9-6 导航搜索

TEAM	MRR	TEAM	MRR
0.594469	TIANWANG_RUN_1	0.193099	APEX_RUN_2
0.486209	ANS_RUN_1	0.192264	APEX_RUN_5
0.466412	ANS_RUN_2	0.190103	APEX_RUN_4
0.416608	MUMIAN2_RUN_1	0.168134	WLL_RUN_1
0.324193	TRS_RUN_2	0.047786	SCUTDB_RUN_1
0.229966	TRS_RUN_1	0.030474	SCUTDB_RUN_2
0.207099	APEX_RUN_1	0.016009	SCUTDB_RUN_4
0.197811	MUMIAN1_RUN_1	0.008830	SCUTDB_RUN_3
0.194185	APEX_RUN_3	0.008772	SCUTDB_RUN_5

之后的工作还包括:维护 CWIRF 网站,大家参与反馈,专业人员检查(包括评判、提出意见和建议),使用数据;评测中涉及的程序都公开,每个参加队可以检查评测结果。

测试集的规模是关键,要发动 IR 相关的研究人员,一起协作来扩大数据集规模。同时,应把评判工作分散开,由大家一起分担,如采取 ftp 上常见的按上载贡献分配下载权利的方式,使用数据资源的必须先提交一定量的评判工作结果;我们来提供基础文档集合和一个开放式的平台,大家可以通过这个平台来参与构建工作。CWIRF 中文 Web 检索评测系列活动,目标始终是构建出大规模可用的数据集,推动中文信息检索研究工作。

### 三、搜索引擎技术评估

前面讲到的评估针对的系统和技术是在原型阶段,本节介绍如何评估实际应用中的搜索引擎,给出其相对优劣次序。

搜索引擎服务者和研究人员不断尝试新的技术,对于提高搜索引擎的服务质量十分重要。商业搜索引擎内部通常会有质量评估,一般不会公开,目前这一方向的工作与研究主要由信息检索领域的研究人员推动。

信息检索可以看做这样的过程和方法,通过检索系统,一个需要信息的用户可以把他的信息需求转换成为对数据集中若干文档的引用,从而找到有用的信息。而“评估”从信息检索这个研究方向创立开始就一直为人们关注。根据评估对象的不同,评估可以分为 6 个级别(Saracevic 1995):工程级,关注系统的效率,具体的评估指标和方法参见(Jian 1991);输入级,关注输入数据的覆盖率;处理级,评估数据处理过程中的算法、技术和方法的效果;输出级,评估结果输出后的交互、反馈等;应用级和社会级,评估系统的应用和对生产率的影响。其中前三个级别的评估以系统为中心,后三个以用户为中心。进行评估有如下几个前提要求:被评估的系统,包括算法和数据;评估准则,如信息检索中的相关性;评估指标,如信息检索中的查准率和查全率;评估指标的获取,如信息检索中的相关性判别;评估方法,包括整个过程的设计与组织。

目前信息检索领域最重要的评估工作由 TREC 组织。TREC 主要在处理级进行检索效果评估,把相关性作为评估准则。相关性是一种复杂的感知和社会现象,它不是简单二元的是否判别,和环境、上下文有密切联系。相关性判别通过专门的评估人员人工判断,在大规模数据集上进行检索系统评估时,这种人工的工作成为最大的开销,并且使得对整个数据集进行完全的相关性判别成为不可能。为了让评估实验可重复,TREC 建立了大规模的评估数据集,包括数据集、查询集和相关结果集。其中相关结果集通过一种称为 Pooling 的方法构造,对每个评估用的查询,从所有参加评估的结果序列中挑选出前一部分,人工判断其相关性,其他文档做不相关处理。在 TREC 的推动下,评估方法的研究得到了很大发展。文献(Zobel 1998)研究表明:

Pooling 方法对系统相对性能评估具有稳定性,但同时查全率被估计过高。文献(Voorhees and Buckley 2002)通过统计实验错误率研究了查询集大小对评估的影响,指出小的查询集会使评估结果具有高错误率;文献(Buckley and Voorhees 2000)研究查询集大小、评估指标对评估结果稳定性的影响,指出对于 Web 环境,适合使用  $P@10$ (前 10 个结果的精度)或  $P@20$  作评估指标,同时需要较多的查询以减少错误率,100 个查询对  $P@20$  较合适。文献(Voorhees 2001)研究多级的相关性判别对评估的影响,指出 Web 环境下对高度相关的评估和一般相关的评估结果不一致,相比更不稳定。文献(Cormack et al. 1998)研究了大规模评估数据集构建的技术,提出了改进的 Pooling 方法。

Web 搜索引擎与传统信息检索有许多不同的特点,这些给传统的信息检索评估带来了新的挑战。Web 搜索处理的数据是整个 Web,即使采用 Pooling 方法,TREC 建立静态评估测试数据集的方法也很难扩展到这样的数据规模。同时 Web 数据还在不断动态变化,对搜索引擎的评估很难建立在同样一个静态的数据集上,如何评估不同数据集上检索系统的质量也是一个新的研究内容。文献(Hawking et al. 1999)研究了 TREC 的信息检索方法在 Web 环境下是否有效,不过实验建立在 TREC 的大规模数据集上,具有一定的偏向性。文献(Hawking et al. 2001)对 11 个搜索引擎进行了评估,指出不同搜索引擎有显著性差异,不同评估指标间高度相关;特别提出了对未来 Web 搜索引擎质量评估,需要针对不同用户的信息需求类型采用不同的评估技术。文献(Singhal and Kaszkiel 2001)详细分析了 TREC Web Track 的评估方法在 Web 环境下的不足,包括查询类型、相关性判别,是按文档为基础还是以用户为中心,按站点为基础,不同数据集上的评估结果是否可以比较等问题。文献(Craswell et al. 2003)是 2003 年 TREC Web Track 的报告,分为主题提取类型和导航类型两种任务模式。主题提取类型以  $P@N$ (前  $N$  个结果的精度)为评估指标,返回结果以站点为单位,以主题相关性和内容质量为评估准则。导航类型以 MRR(第一个正确答案的平均序号倒数)和  $S@10$ (答案出现在前 10 个结果的查询比例)为评估指标。

### 1. 查询类别分析与查询集的构建

用户信息需求千差万别,以不同查询类型表达,这些类型差异对于检索系统十分重要,因为不同类型的需求可以通过不同的检索方法更好的满足,对系统评估也同样如此,不同类型的用户信息需求和查询需要可以采用不同的评估方法。

对用户查询有不同的分类方法。文献(eTesting 2000)从查询语法特征上把用户查询划分为 5 类:自然语言查询、单个查询词的简单查询、多个查询词的简单查询、包含操作符的复杂查询和主页查询。文献(Travis et al. 2001)把用户信息需求分为 3 类:信息型、导航型和事务型。其中信息型是寻找主题相关的文档,也就是传统 TREC 评估的 ad-hoc 任务;导航型是寻找知道名称的站点或主页,对应 TREC 评估

的 homepage/named page 任务;事务型指用户期望找到一个服务入口,需要进行服务访问,如公交线路查询、歌曲下载、产品信息查询等。文献(Kang et al. 2003)也使用同样的 3 个类别,研究了具体的分类识别算法。TREC 的评测不区分信息型和事务型。

我们也可以通过对搜索引擎用户查询日志的统计分析,逆向推断用户的信息需求,得到查询类型的分布。对不同搜索引擎系统用户查询日志大量统计研究都表明,搜索引擎平均用户查询词长度很短,平均长度为 2~3 个词。短查询很难充分表达用户的信息需求,是搜索引擎为提高系统性能面临的一个重要挑战。这也为从查询日志中逆向推断用户信息需求带来了困难。短查询词在不同的上下文环境下存在多义,即使对相同含义,从用户角度也隐含着不同的信息需求。例如,查询“绿茶”,可能是指电影名,也可能是指茶叶;指电影名时,用户可能希望下载或观看,也可能希望了解影评等相关信息,这就分别对应上述事务型和信息型的查询类型。在统计过程中,使用普遍情况最大可能的用户需求作为查询类别的判断准则,如表 9-7 所示。

表 9-7 用户查询信息类别

类别	用户需求	判别准则
导航型	寻找指定名称的站点或主页	指定公司、机构、网站名称的查询
信息型	寻找主题相关的文档页面	页面内容可以满足信息需求的查询;人名、小说、电影查询归到这一类
事务型	寻找到一个服务入口,需要进一步进行服务访问	对软件、图片、音乐等资源的下载查询;信息服务的查询

在一个查询样本中,可对不同类别的查询分别进行筛选,构建查询集。先去除那些可能对评测人员产生困扰的不良查询,如性、暴力方面的查询;再各挑选查询意图明确的若干个查询(如 50 个),构成评估用的查询集。在 Web 搜索环境下,短查询现象普遍,因此很难对挑选出的每个查询补充上确切的查询意图描述,在实验过程中可以统一使用表 9-7 中的评估准则。

## 2. 评估实验的建立与分析

搜索引擎检索质量评估的目标是对不同搜索引擎系统的检索结果评估其相对优劣次序。对单独一个系统的评估,得到的评估指标的得分一般没有实际意义。搜索引擎的搜集和检索两大部分的性能对最终的检索质量都有影响。(Hawking et al. 2001)指出以  $P@N$  为评估指标时,指定结果个数  $N$ ,检索精度随着文档集合大小增长而增长。并且评估对象搜集的网页范围、数量都不相同,这种差异对评估有一定的影响。

可以考虑在实验中采用一种归一化的方法,把查询结果限定在一个固定的集合

内,用来减小不同评估对象的搜集系统差异对检索效果带来的差异。例如,采用 InfoMall(InfoMall 2012)系统容纳的网页集合为基准,对所建立的查询集,用一些工具抓取几个不同搜索引擎的前若干个(如 50 个)检索结果,同时向 InfoMall 系统请求这些结果 URL 的历史网页,当 InfoMall 系统的访问界面返回错误,通知一个 URL 不存在时(HTTP 的 404 错误码),则表明此检索结果不在 InfoMall 网页集合中。

评估实验的基本原则是盲测试性和可重复性。盲测试性是指评估实验应该尽量避免用户的主观偏向性,要求进行相关性评分的评测员不能区分评分对象属于哪一个评估对象,甚至不知道评估对象差别的存在。如在实验中可以把抓取的不同搜索引擎的检索结果保持原始的顺序随机混合,即每个查询的结果序列(如只保留在 InfoMall 中命中的结果)对相同序号上的结果随机排列,合并成一个序列。同时使用统一的摘要提取程序,从网页原文中按查询词提取摘要,以相同的形式展现给评测人员,而且事先不透露评估对象的名称。这一过程可保证实验的盲测试性。

可重复性是一个科学实验的基本要求,TREC 的评估体系在这一方面做得很好,通过构建数据集,包括文档集、查询集和相关结果集,使得评估实验具有可重复性。Web 检索下,这一方法面临的最大问题是数据规模难于扩展,特别是对适合 Web 数据评测规模的文档集,难于有效构造出其中查询对应的相关结果集。TREC 的 Pooling 方法提高了这一工作的效率,但仍然不能适应数据规模的进一步增长。实验中,可以由自愿参加评测工作的评测员,按相关性准则对查询结果的相关性评分。这一过程并不构建一个完全的相关结果集,对一个新的评估对象进行重复实验还必须让评测员重新对变化的查询结果进行相关性评分。

评估实验选择 DCV(Document Cut-off Value)类型的评估指标。检索性能评估通常基于  $P$ (查准率)和  $R$ (查全率),有两类指标,一类是  $P-R$  曲线和根据  $P-R$  曲线计算的平均精度 AVP,这一类指标对不同的查询在同一个查全率上计算平均精度;另一类是 DCV,它使用相同的评测过的文档数量进行归一化,用来表达不同查询结果中用户在同样的浏览代价下的性能。

对大规模数据量的 Web 检索评估,建立基于 Web 的评测环境和招募大量的自愿者参加评估是解决评估的数据规模扩展性问题的一个可能方法。实验中,通过使用相关度评测环境和工具,由志愿评测员对查询集合由各个搜索引擎返回的检索结果进行相关性判别。

对实验结果进行分析:首先进行异常数据清理,然后根据评估指标计算出不同搜索引擎的分值,比较不同搜索引擎的检索质量优劣。由于实验过程存在的随机因素,在结果比较分析中,还必须做统计的显著性检验及实验的错误率分析。对两个不同评估对象,假设根据评估指标计算出的指标得分的差异符合正态分布,可以使用成对数据的  $t$  测试进行显著性检验,在实际应用中,即使正态分布假设并不成立, $t$  测试也基本有效(Hull 1993)。实验可以采用成对数据的  $t$  测试,通常置信度取 95%,对通

过测试的结果认为存在显著性差异,否则没有显著性差异。

## 第五节 小 结

搜索引擎查询结果,动辄几千上万,而用户浏览结果网页,通常不会超过前两页,所以如何做好返回结果相关网页排序是一个挑战。要解决排序问题,需要依赖网页内容分析,网页之间的链接关系分析和查询用户信息分析等技术。目前,搜索引擎公司百花齐放,即使原先不做搜索的门户网站、操作系统等企业也纷纷加入这个市场。如何评价哪个搜索引擎好,需要进行技术评估。

本章首先介绍传统的信息检索领域的向量空间模型排序技术;接着分析网络环境下影响排序的若干新因素,并讨论如何利用 Web 间的链接关系进行相关度排序,介绍了 PageRank 和 HITS 算法;然后给出了考虑这些因素后的一个结果排序的具体实现方案;最后结合 TREC 和天网课题组制作的 CWT100g 测试集来讲解搜索引擎系统质量评估的一般技术与方法。

对于搜索评测的前沿成果,可以参阅高等教育出版社即将于 2012 年 5 月出版的《搜索引擎效果评测——基于用户点击日志分析的方法与技术》,该书作者是何靖和李晓明。



## 下篇 Web 信息资源的组织与应用服务

近十几年来,许多国家陆续开展了长期保存 Web 上的网页与非网页信息的研究,这项工作如同我们保存各种历史文献、资料和档案一样重要,具有很大的社会价值。开展这一研究需要考虑如何对海量 Web 信息资源进行重构与再组织。本篇首先介绍了在这一领域所做的一些研究性工作,包括自行研发的两个大规模 Web 信息仓储系统:存储历史网页的“中国 Web 信息博物馆(Web InfoMall)”和存储 Web 上非网页信息的“中国互联网数字资源财富库藏(CDAL)”,目前这两个系统已存储了海量的 Web 信息资源,并提供了的信息查询服务。随后,介绍了实现中文网页信息组织的两种技术和方法:网页分类与聚类。问答系统是信息检索领域里一个倍受关注并具有广泛发展前景的研究方向,本篇最后介绍开放域问答系统的相关技术方法。

1) 自 2001 年以来,我们开始保存从互联网上搜集来的网页,并重构了网页之间的链接关系。用户可以使用 URL 获取和浏览不同时间点上的历史网页。目前该系统已经收藏中国互联网上出现过的几十亿网页,压缩以后的数据量超过几十太字(TB),并且以平均每月 7000 万网页的速度扩大规模。第十章介绍设计并实现这一大规模历史网页仓储系统(Web InfoMall)的相关技术与方法。

2) 中国互联网数字资源财富库藏(CDAL)是我们实验室建立并维护的综合网络资源库藏系统。2003 年开始建立,目前已拥有近 7.5TB 的存储容量,资源总量约 1.63 万,包括文件数 61 万。CDAL 为开展海量中文数字资源的分类、存储、管理、描述、访问、检索等问题的研究提供了平台,并为国内相关研究提供数据集。第十一章介绍 CDAL 的资源收集、组织分类的策略和方法、存储体系的设计与实现、系统管理与服务方式等。

3) 中文网页的分类与聚类已成为中文 Web 信息处理领域的基础性



工作,如将网页进行自动分类,可以为搜索引擎用户提供目录导航服务,进而提高系统的查准率。第十二章将详细介绍中文网页分类的各种算法,并比较这些方法的优劣。利用网页分类实现搜索引擎的个性化查询服务是一项值得进一步研究的工作,其基本思想是:先对网页和不同用户分别进行分类,之后对两者进行类型匹配,从而实现个性化服务。

4) 问答系统集成知识表示、信息检索、自然语言处理于一体,能更好地满足用户的检索需求,是信息检索系统的一种高级形式。它能用简洁的语言回答用户用自然语言提出的问题。第十三章介绍开放域问答系统的系统框架、涉及的主要技术和评测方法。

## 第十章 大规模 Web 历史网页仓储系统的构建

网页是一种易逝的信息资源,在新网页不断涌现的同时旧网页也在逐步消失。Web 从诞生至今已经有二十余年的时间了,在这期间有无数的网页曾经出现而后消失,甚至有很大一部分已经永久性地消失了。有学者提出,早期的互联网是一段历史真空的时期,因为这个时期的许多信息,如电子邮件、各种网页、数字图片、动画等,在人们还没有意识到它们的历史价值并进行保存的时候,就消失掉了。我们现在还可以找出一百多年前的报纸,但也许再也不能重现数年前一个热门网站的面貌了,即使我们仍然依稀记得它的模样。因此,持续收集并长期保存 Web 网页具有重要的史料价值和社会意义。

自 1996 年以来,国外相继有一些组织和政府机构开始对互联网资源进行收集和保存工作,规模较大的是美国的 Internet Archive 系统。在国内,北京大学网络实验室在国家 973 和 985 项目支持下,从 2001 年开始了国内 Web 信息的收集与保存工作,构建了一个大规模的 Web 历史网页仓储系统——“中国 Web 信息博物馆”(http://www.infomall.cn),简记为 Web InfoMall。该系统收集了 2001 年以来国内网站在不同时间点上的网页信息,目前已有 40 多亿网页,并且以平均每月 7000 万网页的速度扩大规模。为了与当前互联网上实际存在的网页相区别,我们称这些定期收集保存下来的网页为“历史网页”。

本章主要介绍 Web InfoMall 系统构建的技术与方法。其中,第一节介绍国外网页保存的现状,包括 Internet Archive、PANDORA 等大型网页保存系统。第二节介绍了 Web InfoMall 的功能及系统体系结构。第三节给出了 Web InfoMall 存储历史网页的具体方法。第四节介绍了数据访问的技术与方法。第五节给出了保存 Web 网页的一种格式,该格式也可作为网页抓取时的存放格式。

### 第一节 国外 Web 历史网页保存现状

从 20 世纪 90 年代中期开始,国外的一些机构就开始尝试 Web 上信息的保存工作。其中大多数是有选择性的,针对特定国家域名、主题或个别网站的居多。开始最早且规模最大的是美国的 Internet Archive,它从 1996 年开始对整个世界范围内 Web 信息进行周期性的全面搜集。随后,越来越多的国家开展了形式类似的 Web 保存工作。

从网页的搜集模式来划分,Web 的保存工作可分为两个大的类别:非选择性搜集模式和选择性搜集模式。前者对公开发布的网页进行搜集和存储,如 Internet

Archive, 以及只对本国域名进行收集的瑞典皇家图书馆的 Kulturarw 项目、奥地利的 On-Line Archive 项目等。选择性收集模式则是依据各自的设定目标, 精心挑选网站然后予以保存, 典型代表是澳大利亚国家图书馆的 PANDORA 项目。下面对这些 Web 保存项目做简要介绍。

### 一、Internet Archive

美国的 Internet Archive(<http://www.archive.org>) 是美国最早保存 Web 信息的机构之一, 它从 1996 年开始搜集并保存世界各国的网页。它的目标是建设一个 Internet 图书馆, 为研究人员、历史学家和其他学者提供永久性的数字资源的历史存档。

Internet Archive 保存的数据量十分惊人, 至今已拥有超过 200TB 经过压缩的数字资源, 其中最主要的数字资源是网页, 保存有 850 亿个覆盖全世界范围的网页存档。作为比较, 美国国会图书馆馆藏文本资料, 大约相当于 20TB 的数据(Kahle 1997)。

从 1999 年开始, Internet Archive 还搜集其他类型的数字资源, 包括音频、视频和文档资料。除了接受捐献外, 还和一些图书馆合作搜集包括“中美百万册图书项目”在内的众多电子书籍和各种电影、录音等。不过这些资源和网页相比, 还相对较少。

2001 年, Internet Archive 提供了一个称为“Way Back Machine”的 Web 服务, 它使得人们可以通过浏览器来访问 Internet Archive 的数据资源, 即用户根据 URL 查询网页存档并浏览网页。此外, 它提供数据共享, 通过申请, 可以直接复制得到这些数据资源。

Internet Archive 的数据存储在 PC 集群上, 存储采用的是 IDE 硬盘, 操作系统是 FreeBSD 或 Linux 操作系统。按照 Sata(Sata 2002) 的说法, 每太字节数据的硬件花费大约是 3000 美元。

### 二、PANDORA

澳大利亚国家图书馆的 PANDORA 项目(Preserving and Accessing Networked DOcumentary Resources of Australia, <http://pandora.nla.gov.au>) 始于 1996 年, 它只对澳大利亚国内一些网站进行有选择性的定期搜集, 大致搜集了 1000 多个网站。与此同时, 它还设计了一个网站选择指导原则(<http://pandora.nla.gov.au/selectionguidelines.html>)。澳大利亚国家图书馆认为(Law 2001), 搜集和保存电子出版资料是一件复杂、耗时并且昂贵的工作, 因此需要经过选择, 把资源集中在那些现在和未来都有研究价值的出版资料上。一旦网站被选中并且得到网站拥有者的同意, 就会采用搜集软件进行采集。

PANDORA 对收集的网站分类整理, 由于经过人工挑选, 具有很强的代表性, 并提供友好的导航界面。除了网站选择, 它还考虑了一些其他问题, 如采集的频率取决

于实际需要,有些可能只采集一次,而有的则需要每周搜集。此外还有采集的深度等问题,假如网站很大,则只采集部分网页数据。

### 三、其他相关 Web 保存项目

除了上面介绍的 Internet Archive 和 PANDORA,还有许多其他的 Web 保存项目。这些项目多是由各国国家图书馆发起的,下面介绍一些比较有名的项目。

采用选择式信息收集模式进行 Web 保存工作的项目,主要有:

1) 美国国会图书馆 Minerva 项目(Mapping the Internet the Electronic Resources Virtual Archive,<http://www.loc.gov/minerva>)。它始于 2001 年,搜集和保存可以公开访问的 Web 资源。Minerva 的目的是尝试有选择性地搜集保存 Web 站点的课题,最终为大规模的 Web 保存系统的开发提供实际操作经验。Minerva 还与 Internet Archive 合作,针对历史事件制作了“September 11 Web Archive”、“Election 2002 Web Archive”等专题性网站向公众开放。Minerva 认为,搜集保存大规模网站的系统应该能做到大部分工作是自动化处理完成的,并且由图书馆管理人员负责维护。

2) 大不列颠图书馆的 Britain on the Web 项目。选择了英国 100 个有代表性的站点进行保存,这些站点包括了新闻价值高和题材面广的网站。

采用非选择性搜集模式进行 Web 保存工作的项目,主要有:

1) 瑞典皇家图书馆 Kulturarw 项目(<http://www.kb.se/kw3>)。自 1997 年开始搜集数据,使用一种称为联合收割机(Combine)的软件,预计每年搜集 2~3TB Web 信息,规模远小于 Internet Archive,而且这些数据资源不能公开访问。

2) 芬兰赫尔辛基大学图书馆(芬兰国家图书馆)启动的一个类似 Kulturarw 的项目——EVA(<http://www.lib.helsinki.fi/eva/english.html>)。该项目仅搜集局限于 .fi 域名内的网站。

3) 欧洲的 NEDLIB 项目(Networked European Deposit Library, <http://www.kb.nl/coop/nedlib>)。该项目有 8 个欧洲国家图书馆参与,它们的搜集系统(NEDLIB harvester)从 2001 年开始工作。

4) 奥地利的 On-Line Archive 项目(AOLA, <http://www.ifs.tuwien.ac.at/~aola>),由奥地利国家图书馆发起,也使用联合收割机(Combine)搜集系统。搜集的对象是与奥地利文化和传承相关的网站。

其他相似的 Web 保存项目还有很多,在此不一一介绍,列举如下。

- 捷克 WebArchiv(<http://webarchiv.nkp.cz>)。
- 丹麦 netarchive.dk(<http://www.netarchive.dk>)。
- 德国 deposit.ddb.de(<http://deposit.ddb.de>)。
- 日本 Warp(<http://warp.ndl.jp>)。
- 美国 NDIIP(<http://www.digitalpreservation.gov>)。
- 挪威 Paradigma([http://www.nb.no/paradigma/eng\\_index.html](http://www.nb.no/paradigma/eng_index.html))。

- 荷兰 archiPOL(<http://www.archipol.nl>)。
- 英国 UK Central Government Web Archive(<http://www.pro.gov.uk/web-archive>)。
- 美国 CyberCemetery(<http://govinfo.library.unt.edu>)。

综上所述, Web 信息保存得到了越来越多的国家和机构的重视。但保存这些信息是一项艰难的任务。这不仅因为 Web 的规模非常庞大,而且 Web 的许多内在特性使得保存工作变得更加困难,主要包括如下几个方面。

1) 非集中式的组织结构。Web 是以一种非集中式的自由的模式发展起来的。没有任何机构和组织对整个 Web 负责,也没有相应的约束法律对网站的保存策略制定过标准。网站的管理维护完全取决于网站的所有者。

2) Web 的动态特性。Web 处在一种永不停息的演化中,随着时间的变化,网页、网站甚至整个域名不停地经历着产生、消亡、更新、重构、取代的过程。网页的生命周期,有的仅有几个小时就消亡,但大部分网页时无法准确地预测消亡时间,因此很难制定合理的网页搜集策略。

3) Web 技术的演变。尽管 Web 基本技术标准和协议保持着相对的稳定,但是技术更新层出不穷。许多网站的维护方式已经发生根本性变化,如采用动态的数据库来发布网页。如果没有详细的元数据描述和相应的软件、数据库结构信息,对这些网站的复制存储将变得非常困难。

4) 法律问题。包括版权问题、缺乏合法的存档机制、可信度问题,以及隐私和秘密的保护等。

就目前的现状来看, Web 保存工作尚处在一个较为初级的阶段。大多数工作的重点是集中在网页的搜集这个部分,对于历史网页通常仅提供简单的数据访问,例如给定一个 URL,返回保存的历史网页。而对于历史网页的分析、挖掘,以及历史网页的内容检索等这些较高层次的工作,则很少涉及,即使有的曾经作过一些尝试,也并没有成功实施。例如, Internet Archive 在 2005 年建立过一个历史网页全文检索的测试版,因效果不好而没能维持下去。

此外,学术界对该领域的研究大多集中在网页的搜集和存储方面,在历史网页的分析、挖掘、展现、检索这些方面涉及很少。可能的原因有三:一是 Web 保存项目的发起人很多不是计算机方面的专业人员,他们通常更关心一些非技术性的问题;二是 Web 保存属于公益性事业,难以被用来获利,缺少相应的利益驱动;三是由于数据量十分庞大,研究难度较大。正因为如此,在这个领域尚存在大量的课题有待于人们去研究和解决。

## 第二节 中国 Web 信息博物馆的系统设计

2001 年,北京大学天网组着手构建中国 Web 信息博物馆,目前已建设成为一个

大规模可扩展的历史网页仓储系统。它可以增量地搜集中国 Web 页面,至 2011 年 12 月,Web InfoMall 已经搜集有超过 40 亿的历史网页,并以每月大约 7000 万网页(未压缩是大小约 2.4TB)的数量递增。

“网页历史回放”是 Web InfoMall 所提供的一项基本应用服务。利用该服务,用户可以浏览 Web InfoMall 所保存的历史网页。例如,用户可以浏览过去某一个日期,如 2005 年 7 月 1 日新浪网的主页;还可以顺着历史网页上的链接继续点击,查看 2005 年 7 月 1 日左右系统保存的其他历史网页。

### 一、Web InfoMall 的设计目标

从设计理念上讲,Web InfoMall 2.0 是一个大规模的 Web 历史网页仓储系统,使用较低成本的硬件和软件系统来实现大规模历史网页的存储访问,并为高层次的应用开发提供接口和支持;从设计目标上讲,它的目标是完成如下任务:

1) 持续保存中国范围内的所有网页。

2) 再现历史网页,建设一个历史网页回放和展示的网站,使人们可以方便地访问过去的网页。

3) Web 历史信息挖掘与检索,这需要研究海量数据访问技术和存储结构来支持海量历史网页的挖掘和检索。

4) 考虑到 Web 历史信息开发利用将会是一个长期探索的过程,因此 Web InfoMall 2.0 也应是一个 Web 历史信息开发平台,它集成各种工具和数据,使各种新的思想和技术能够很快地得到实施、测试和验证。

基于这样的目标和理念,就要对 Web InfoMall 的仓储系统提出客观的性能和功能上的要求。如果一个 Web 仓储系统能够提供某种功能上的有效访问和处理能力,我们称它提供了这方面的视图。从概念上分析,Web InfoMall 的仓储系统应当提供如下几种视图。

1) 网页视图:实现网页数据获取及展示,如按 ID、URL 等访问网页对象。

2) 文档视图:实现文本检索,例如词的倒排索引,需要记录词的位置、视觉效果、页面分块等信息。

3) 链接视图:可用于链接分析,如提供网页链出关系和链入关系的信息。

4) 属性视图:可用于历史网页挖掘与检索挖掘,如对链接分析和网页分析挖掘所得到的信息(分类、权值、时间、地点、人物等)予以组织存储并提供有效访问。

### 二、Web InfoMall 的体系结构

Web InfoMall 的基本定位有两点:一是系统要具有良好的可扩展性,二是采用较低配置的硬件设施,因为有限的资源主要投入到了存储设备即磁盘系统上,内存和计算资源(CPU)相对有限,这就对系统的开发提出更高的要求,需要更多地采用优化措施,如压缩技术等。

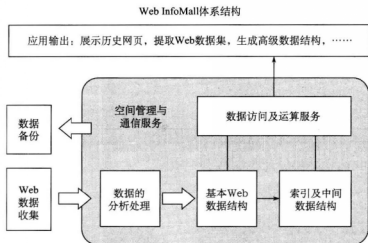


图 10-1 Web InfoMall 体系结构

本着量力而为、逐步推进的工作思路，我们提出的 Web InfoMall 的系统结构，如图 10-1 所示。图中的灰色背景是 Web InfoMall 的核心部分——仓储系统。它是一个在线的服务系统，可接收来自 Web 收集系统的数据，具有输出应用接口或者输出特定的数据，为保证数据的安全，还需要定期对数据进行备份。Web InfoMall 系统的各个组成要素描述如下。

1) Web 数据收集：由一个 Web 搜集器（当前是天网搜集系统）负责从互联网上搜集 Web 数据，包括 HTML、PDF、FLASH、图片等，这里统称为网页。Web 数据搜集也可以来源于其他途径，如与其他网页仓储系统进行数据交换、网站赠与/上缴。

2) 数据备份：历史网页数据长期保存与异地备份。需要注意的是，数据应当存储为适合长期保存的格式，不能依赖特定的硬件和系统。

3) 数据的分析处理：网页数据经过分析处理，存储为内部数据结构，可分为两类：一类是基本 Web 数据结构，是对输入数据的第一次分析处理后的结果；另一类是索引及中间数据结构，它们在基本 Web 数据结构基础上生成。

4) 基本 Web 数据结构：指包括原始网页数据及基本元数据。其中，一些元数据可以从搜集器获得，如 URL、IP 地址、收集时间等，另一些则需要进一步分析提取，如编码、语言、摘要等。一般来说，基本 Web 数据结构一旦生成就不再修改，适合采用紧凑的方式进行存储。

5) 索引及中间数据结构：在基本 Web 数据结构的基础上，为了加速访问而生成的数据结构及进一步分析挖掘得到的中间数据结构，如 URL 索引、文本索引、链接图结构等。这类数据结构用来对上层应用提供数据支持，需要具有高速访问和运算的能力。它们可以从基本 Web 数据结构重新生成，可能需要频繁的创建、修改及删除操作，因此对于空间管理的要求较高。

6) 数据访问及运算服务:这是系统对外提供应用输出的接口,它读取下层的数据结构,进行检索、匹配及交集、合并、排序等运算操作,然后将结果输出给外部应用程序。

7) 空间管理与通信服务:这是系统内部为简化开发而建设的软件环境,主要职责是进行空间管理和提供通信服务。它完全可能是一个非常复杂和庞大的子系统,这主要取决于整体系统的规模。

8) 应用输出:系统的输出可能是针对某种应用服务的,也可能是某种高级数据结构,这种高级数据结构可用来进一步分析挖掘或者实现新的应用服务。系统能够提供什么样的应用输出取决于它所实现的内部数据结构。

将 Web InfoMall 需要存储的数据划分为基本 Web 数据结构和索引及中间数据结构,是基于两个方面的因素,一方面基本 Web 数据结构(主要是原始网页数据)的存储及访问问题是首先需要解决的核心课题,而其他的数据则可以在这个数据的基础上重建得到;另一方面,基本数据结构是一种静态的数据,一旦生成就很少需要再更新,因此可以更多地进行一些优化和紧凑存储,而其他数据则常常是动态的,需要频繁地更新。

索引及中间数据结构本身也包含了很多类型的数据结构,如 URL 索引、倒排索引、链接图等,很难使用同一种方法来存储。特别是,需要应用压缩技术等来优化存储,而不同的数据其压缩方法是大相径庭的。一些系统如 IBM 的 WebFountain (Gruhl et al. 2004),以及 Google 的 BigTable(Fay Chang 2006),它们倾向于一个存储系统可以存储很多种不同的数据类型,并且也各自取得了很好的效果。不过这样做,存储代价比较高昂,两者都需要投入大量硬件资源,包括数以千计的存储机器。

综上所述,Web InfoMall 仓储系统的设计和实现是一个需要逐步完善的过程,目前的设计为将来系统的扩展留下空间。

### 第三节 历史网页的存储

基本 Web 数据结构由原始网页及其元数据组合而成,其中元数据附加在网页上,所占比例很小,因此基本 Web 数据结构的存储问题可以理解为网页的存储问题。简单起见,可以直接使用“网页”来表示“基本 Web 数据结构”。

Web InfoMall 的仓储系统是可扩展的,即系统容量可以随着机器的增加而线性增加,同时只有少量性能上的下降。为实现可扩展性,从设计初期就应分析机器数量增多时系统性能的变化问题。如每台存储机器有它独立的服务系统,可以独立提供数据服务,并且可以很容易加入或者从存储系统中脱离。在这些机器中,有一台存储机器作为主存储机器,它负责周期性地向所有的存储机器查询并收集数据资源信息,为避免成为系统的瓶颈,它的作用主要是区分哪些数据资源存放在哪台机器上,并不参与实际的数据交换。



## 一、数据的组织

网页数据的访问模式有两种,一是顺序访问模式,即按照存储顺序读取网页,这种读取的速度很快,通常每秒可多达数千个网页,其速度的大小主要取决于不同的磁盘系统。二是随机访问模式,即按照随机顺序读取网页,由于网页存储在磁盘上,如果是完全随机的话,意味着每次读取需要访问一次磁盘的不同位置,所以速度是非常慢的,往往每秒只能读到 100 个左右的网页。

随机访问速度很难有实质性的提升。这是因为磁盘采用的是一种机械调度的方式,它与磁盘的转速和机械臂的移动速度有关,由于是机械的行为,不可能做得很快。因此,除非有革命性的技术进步,随机访问速度难以得到大幅度的提升。尽管对于一些特定的应用,使用内存缓存可以大幅提高性能,但在大多数情况下,对于大规模数据处理来说,内存缓存作用十分有限。所以,对数据的读取通常采用顺序访问模式。

如何把网页分配到各个存储机器(或存储位置)上。直观地看,历史网页有两种组织方式。一种是按 URL 顺序来组织网页,这可以理解为空间顺序。例如很多 Web 存储系统采用的都是依据网页的 URL 地址,将网页散列到各个机器上,同一站点的网页可以分配到同一机器、同一目录,或者同一文件。这样做有一定的好处,如果要读取某一网站或某一 URL 范围,可以很快,但是有空间管理和访问上的问题:一是很难准确分配存储空间,势必造成空间上的浪费;二是原数据集(同一时间段)的重新合并非常缓慢。另一种组织方式则是选择按搜集时间来顺序组织网页,即时间顺序。也就是将同一时间段的网页存储在一起,这样读取某一时间段的历史网页集时,可以做得很快。

我们认为对于历史网页的挖掘和分析而言,时间上的相关常常要比空间上的相关更有意义,如同一新闻网站不同时期的网页之间通常没有什么联系,而同一时刻不同新闻网站的网页却常常反映同一时期的历史事件。另外这种方式存储较为紧凑,不会造成空间上的浪费。因此,我们主要采用依照时间顺序来组织历史网页,同时也留下一定的改进空间。大致上讲,就是在一个适当的时间单位(如一个月)将历史网页按时间顺序来存储,而在这个时间单位内部,则可以更灵活一点。

Web InfoMall 采用的是增量搜集方式,它在时间上是持续不断的,它的搜集器源源不断地从 Web 搜集网页,然后加入到系统中来,这个过程可以是无限延伸的,如图 10-2 所示。将这个连续的数据流以月份为单位分割批次,分别予以存储。显然,不能认为系统的存储容量是无限大的,所以当网页数量大于系统容量时,就需要将一部分网页下线离开服务。因此数据应该能方便地加入和离开。此时可以将批次作为一个数据单位进行操作,批次之间相对独立,一个批次的加入和离开,不会对其他批次造成影响。

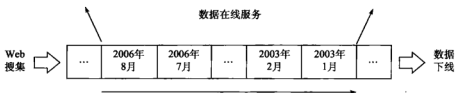


图 10-2 网页数据的分割

## 二、存储结构

在将网页加入存储系统之前,首先需要对网页做一些分析,从网页中提取出一些基本的元数据,主要包括:

- 1) 网页的 content-Type、Last-Modified-Time 等 HTTP 属性。
- 2) 网页的编码:UTF-8、GB18030、Big5 等,有时网页中并不包含相应的编码信息,甚至有的提供了错误的编码信息,因此需要运行一些测试函数。
- 3) 网页的语言属性:简体中文、繁体等,同样也需要运行一些测试函数。
- 4) 网页的 MD5 摘要。

这些提取出来的元数据,与经过压缩的原始网页数据、URL、IP 地址、校验码等,组合成一个记录,作为数据读写的基本单元。之后,很多个记录聚集在一起,形成一个文件。一定数量的文件(一般是同一个月)存储在一起,形成一个批次。

作为标识,每个批次使用一个不超过 8 字节的字符串 BATCH 予以命名。通常采用这个批次的搜集时间来命名,如“200703”,代表 2007 年 3 月,即该批数据是 2007 年 3 月搜集的。同时,对批次内部的每个记录赋予一个 32 位的整数序号 RECNO,这通常对应它的存储顺序。由此,Web InfoMall 里的每个记录就可以用一个 BATCH;RECNO 来唯一标识,统称为 PageID。

在将历史网页集从逻辑上分割为批次以后,就需要将它们存储到各台机器的实际磁盘上。随后面临的问题是如何提高空间利用率?由于一台存储机器通常很难恰好容纳一个批次的数据,因此直接将批次作为网页的物理存储单位就会造成很大的空间浪费。为了避免这种情况,使用更加灵活的 Depot 物理存储单位来存储网页,如图 10-3 所示,一个批次的历史网页可以存储到多个 Depot 中,而同时一个 Depot 也可以存储多个批次的历史网页。此外,一个 Depot 可以很容易地分割成多个小的 Depot,并且多个小的 Depot 也可以很容易地合并成为一个大的 Depot。我们开发了相应的工具来完成这些操作。

一台存储机器包含多少个 Depot 主要取决磁盘的分区情况。每台机器只生成一个磁盘分区是一种较为简单的方法,但这也有很大的风险:一个磁盘的损坏会造成整个分区数据的丢失(这时就需要从备份中恢复数据),因此并不倾向于一台存储机器仅生成一个分区。

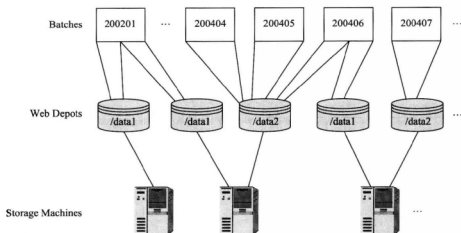


图 10-3 Web InfoMall 的存储结构

Depot 中除了存储网页数据外,也存储了一些其他数据,如 URL 索引、日志和一些用于管理的元数据等。基于这些网页数据和它的 URL 索引,每个 Depot 会独立运行一个数据服务,单独提供服务。如果一台机器包含了多个 Depot 数据服务,我们会在这台机器上建立一个数据代理服务,合成并代表将多个 Depot 对外提供服务。

这里可能存在一个问题,即随着系统的进一步开发,Depot 里面需要存储的数据逐渐增多,以致超出了原来的预留空间,该怎么处理呢?方法是:重新生成这些 Depot,并且是整个机器地操作。具体的,将旧机器里的数据按照新的需求重新生成到空白的机器里,然后这台旧机器又可以成为一台空白的机器导入其他机器的数据。这样做的好处是:可以避免预留太多不必要的磁盘空间,造成浪费;由于网页数据是一种静态的数据,一旦生成少有更新的必要,因此这样的操作并不是很频繁。

### 三、数据管理与压缩

当前,数据库系统被广泛应用于各种数据的管理。但采用数据库系统来管理历史网页数据并不太合适:一方面,历史网页的数据规模非常大,高达数十亿甚至更多,普通的数据库系统不能满足如此规模的数据存储需求,且高端的商用数据库系统十分昂贵;另一方面,历史网页的存储不同于通用的数据存储,它对数据访问的速度要求很高,因而需要针对特定访问要求在各个技术层面上进行优化,采用现有的数据库系统无法做到这一点。

因此,我们自行开发了底层的数据存储系统,并引入各种优化措施以获得更好的性能且占用较少的磁盘空间。此外,搜集来的历史网页很少被删除或者更新,因此历史网页存储系统可以设计得比普通的数据库系统简单得多。例如,当删除一个网页时,系统并不是真的将它从磁盘上删除并回收它所占用的空间,而是仅仅作了一个标

记表示该网页已被删除。因此,在系统中对网页数据的频繁修改是不允许的,如果需要更新网页数据,最好的方法是重新生成整个文件。

当前采用 BerkelyDB(<http://www.sleepycat.com>)来存储网页,它是一个开放源码的底层数据操作函数库。其他的一些数据则存储在结构化的文件里。为节省存储空间,通常需要对数据进行压缩,有两种解决方案:一种是每个网页在写入文件之前先进行压缩,常用的是 zlib 库(<http://www.gzip.org/zlib>);另一种是对整个文件进行压缩。整个文件压缩虽然可以获得更好的压缩效果,但由于通常只访问单个网页,而数据的压缩和解压缩又比较消耗 CPU 时间,因此对每个网页单独压缩效果更好。

历史网页的大规模数据导致了对存储资源的巨大需求。如果能在网页的压缩上取得一点进步,就可以节省大量的存储空间,从而保存更多的网页。下面简单介绍一些网页压缩方面的方法和成果。

直观地说,网页压缩最简单的方法是采用通用的文本压缩算法对单个网页进行压缩,当前有两个流行的开源文本压缩算法,gzip(Deutsch 1996)和 bzip(Burrows et al. 1994),在历史网页数据上测试表明,gzip 的压缩比大约为 3.5 : 1,而 bzip 的压缩比大约为 4.5 : 1。从压缩比上看 bzip 的压缩效率要高一点,但是,gzip 的速度要比 bzip 快很多,因此在大规模数据处理中,采用了 gzip 对单个网页进行压缩。

除对单个网页文档进行压缩以外,还可以利用历史网页集中网页之间的一些特点进行数据压缩,主要包括三点。

- 1) 很多网页完全相同,如镜像、Linux 文档等。
- 2) 同一个 URL 在不同日期搜集的版本,一般都很相似。
- 3) 同一位置的网页(同一网站、同一目录)通常比较相似。

针对特点 1),可以采取完全相同的网页只保留一份的存档,称为复制压缩。针对特点 2)、3),可以只存储相近文档的差异部分,称为差异压缩。

考虑到随机访问速度非常慢,这两种方法都面临着较大的问题。以复制压缩为例,有些系统用的是计算文档内容的哈希值为 KEY(如 MD5、SHA)来维护一个文档内容的数据库。当遇见新文档时,需要一次磁盘寻址操作,判断该文档是否已经存在,此时,顺序访问和随机访问一样慢,非常不利于大规模数据处理。差异压缩同样面临着这样的问题,一个网页在存储时引用了其他网页,只存储自己与被引用网页的差异部分,那么在读取该网页时,就需要同时读取被引用的网页,如果两者存储在磁盘的不同部位,就会导致两次磁盘寻址操作。这会导致本来可以很快的顺序访问变得很慢。

因此,基于访问速度和空间分配的考虑,复制压缩或差异压缩在小范围内有一定可行性,但在整个系统范围内使用是不可行的。于是提出一个引用压缩的设想,具体如图 10-4 所示。在一个批次内部进行一定的规整重组,使相近的网页聚集在一起,然后在同一文件内部,使用窗口长度为  $w$  的引用差异压缩,即网页最多只能引用位

于它之前的第  $w$  个网页,存储差异的部分。这样,在内存中保持一个长度为  $w$  的网页窗口可以实现高速的顺序访问,同时对于随机访问,如果从该网页之前的第  $w$  个记录开始读取(磁盘可以一次读取较大的数据块),那么数据随机读取的效率也不会下降。

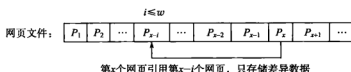


图 10-4 网页的引用压缩示意图

使用该方法的一个重要技术前提是需要一种适合较大规模网页的相似性检测方法,将相似的网页聚集到一起。本书第七章所述的网页消重算法可用于此项检测。

#### 四、存储性能

系统的存储性能与硬件设施有关,特别是磁盘的读写速度和 CPU 的运算能力。因此,SCSI 硬盘的表现要优于 IDE 的硬盘。表 10-1 为在一台使用 IDE 硬盘的存储机器上的测试结果。

表 10-1 显示,压缩使得网页的存储变慢了一些,但是磁盘的占用也减少了。因此,压缩的优点是:网页总体压缩比达到了 3.5 : 1;一些操作的性能因此得到显著提高,如将网页从一台机器复制到另一台机器时性能明显提升,这在数据读取和系统维护中会经常用到。

表 10-1 网页存储性能(个/秒)

直接写入	提取元数据后写入	提取元数据并压缩	复制压缩数据	复制未压缩数据
1350	295	220	3660	950

注:①网页未经元数据提取和压缩直接写入;②网页进行了元数据提取但在写入前未经压缩;③网页在写入前提取元数据并压缩;④压缩格式的网页从一台机器复制到另一台机器;⑤未经压缩的网页从一台机器复制到另一台机器。

### 第四节 数据访问

Web InfoMall 的网页仓储系统提供两种基本的数据访问机制,一种是根据网页的 PageID 来读取网页,另一种是根据 URL 来读取网页。显然,遍历所有网页来寻找所需网页是不合适的,一种较好的办法是分别构建两种索引来获得有效的数据访问,同时构建数据服务来为上层应用开发提供便利。

## 一、PageID 的索引

PageID(BATCH;RECNO)用于唯一地标识每一个网页记录。PageID 的索引是在网页写入存储系统时同时创建的。一个 PageID 由一个批次名(字符串)和一个整数序号组成,因此当一个网页被加入到一个批次时,它会被顺序地赋予一个从 1 开始的整数序号(RECNO),同时需要一个数组来记录整数序号到网页存储地址的映射。这个过程的实现方法简易而且多样,我们采用的方法是:每个存储网页的文件在文件内对网页记录从 1 开始予以编号,然后在批次内维护一个数组,该数组记录了文件内编号与批次内全局整数序号(RECNO)的对应关系。除此之外,还需要建立一个批次名(BATCH)的全局索引。这个索引位于主存储机器,并且是动态维护的。当一个批次被创建时,它必须将自己的批次名(BATCH)注册到这个索引上。主存储机器会定期向所有存储机器发出查询请求,以检查该批次数据是否在线。

一台存储机器可能包含多个批次的数据,其中有些批次可能是不完整的。因此当一台存储机器启动数据服务时,它需要向主存储机器上的全局索引注册它所包含的批次名(BATCH)及整数序号(RECNO)范围。此外,多台存储机器可以包含同一部分的数据,主存储机器在接到该部分数据的查询请求时,会从中选择一台作为返回结果。

## 二、URL 的索引

虽然 PageID 的索引容易实现,但人们通常更愿意使用 URL 来访问历史网页。相比之下,URL 索引的创建要复杂得多。在 Web InfoMall 系统中,每个 Depot 会建立一个 URL 索引,但与 PageID 的索引不同,URL 的索引并不是将 URL 直接映射到网页的存储地址,而是映射到网页的 PageID,这是因为网页的存储地址可能会在系统维护(如数据移动)时发生改变,而 PageID 则是固定不变的。

目前,使用较多的索引结构有如下几种。

1) 二分搜索树(Binary Search Tree):最简单的索引结构,每个树节点存储一个数据和两个指针指向左、右两个子节点,变种有 AVL 树、红-黑树、splay 树等。

2) B 树:每个树节点存储最多  $2d$  个数据和  $2d+1$  个指针指向子节点,变种有 B<sup>+</sup> 树和 B<sup>+</sup> 树等,比较适合磁盘存储,数据更新和读取所需的磁盘访问操作较少。

3) Trie 树:每个节点是一个指针数组指向子节点,指针的位置表示它是什么字符,由于不需要比较操作,所以访问速度较快,有一些压缩变种,但它耗费大量空间。

4) T 树:一种将 AVL 树和 B 树结合发展而来的索引结构,常用于内存数据库,在不需要访问磁盘的情况下效率较好。

5) 哈希表(Hash-Table):是速度较快的一种索引结构,但它是一种无序的索引,且难以压缩。

当前使用 B<sup>+</sup> 树(Comer 1979)来构建 URL 索引。对于数量巨大的网页,URL

索引在构建时需要以较快的速度来完成,磁盘性能就显得尤为重要。举一个例子,有 1 亿网页需要索引,如果内存太小,大部分索引数据放置在磁盘上,这将使得每次索引访问一次磁盘,一般每秒只能索引大约 100 个网页,那么整个过程就需要耗时 11.5 天。相反地,如果索引机器装备有足够大的内存来将全部索引数据完整地载入内存,那么索引的速度就会快得多。

由于北京大学天网的存储机器并没有装备足够大的内存,因此采用了一种两个阶段的索引过程。在第一阶段,将网页集分割为足够多的小网页集,使得每个小网页集的索引数据可以完全载入内存中。这个时候对每个小网页集单独作索引,并且依据一个哈希函数对 URL 的 host 部分进行计算,将整个索引数据分成  $N$  份。在第二阶段,将所有小网页集的索引数据合并,每一次只合并一份,一共合并  $N$  次,以确保该过程可以完全在内存中进行。

由于在第一步的索引过程中,已经对 URL 进行了排序,因此第二步索引的合并也可以在较少内存下很快完成:每次读取最小的 URL,按顺序将 URL 插入  $B^+$  树,只要维持  $B^+$  树的非叶节点滞留在内存中,就可以按顺序将索引数据写入磁盘。

使用这种两个阶段的索引方法,URL 索引数据就可以很快完成。通常,在一台存储机器上每秒种可以完成超过 3000 个网页的索引。

北京大学天网的增量搜集系统在持续不断地抓取网页,因此对于接纳最新网页的 Depot 来说,存在一个增量索引的过程。对于每个新来的网页,如果将其直接索引到完整的 URL 索引中,效率会比较低,因此方法是在内存中建立一个小的 URL 索引。新来的 URL 先存入这个小的内存索引中,在整个索引数据达到一定程度后再写入磁盘中,然后在内存中重新建立新的内存索引,最后在小的 URL 索引达到一定数量后,执行一个合并操作(如同前述的第二步操作)。

需要注意的是  $B^+$  树索引的空间效率:理论上讲,树节点的填充率在 50%~100%,如果不加处理的话,实际填充率一般在 80% 以下,就是说每个树节点有 20% 的空间浪费。因此,在存储之前要进行排序处理,使得填充率可以达到 90% 以上。URL 平均长度约为 65 字节,再加上  $B^+$  树结构本身占用一部分空间,每条记录大约需要 12 字节,在填充率为 90% 的情况下,占用空间大约为 URL 总长度的 3/4,再加上索引载荷(即该 URL 的 PageID 及其他索引信息,以平均 10 字节计算),可以得出:1 亿个 URL 的索引,约需要占用 10GB 的存储空间。

由此看出, $B^+$  树索引的空间效率不是很好,需要占用较大的空间。还有一种可用的压缩方法:在一个节点内部 URL 仅存储它与前一 URL 的不同部分,这大约可以压缩 50% 的存储空间,但它会对存取性能造成一定的影响。

### 三、数据服务

从程序开发的角度,让高层应用程序直接读取底层的数据结构并不合适。因此可以为每台存储机器构建数据服务,进而为应用程序的开发提供便利。

数据服务的功能是接受数据访问的请求并答复,它被设计为多线程的,可以接受多个请求。为了简化应用程序的开发,它提供了一系列的应用程序接口(API)。当前提供的 API 主要有:

- 1) 根据 PageID 读取网页。
- 2) 根据 URL 和抓取时间读取网页。
- 3) 查询 URL 对应的存档网页数,可以指定一个查询时间范围,这是因为持续不断地抓取网页,一个 URL 可能对应多个存档。
- 4) 根据 PageID 查询对应的 URL、抓取时间及其他元数据。

通过使用这些 API,应用程序可以方便地读取网页数据,从而屏蔽底层的技术细节。

由于每台存储机器上存在着一个数据服务,因而当用户需要查询一个 URL 的所有网页存档时,它必须同时向所有存储机器发出查询请求。然而,Web InfoMall 的存储机器分布在不同的位置,有的甚至在不同的机房,由此我们开发了一个基于 UDP 的消息通信服务来屏蔽它们的地理位置:用户只需要向主存储机器发出查询请求,这个请求会被传递到所有的存储机器,而所有存储机器的应答则会沿着相应的路径返回,主存储机器在对查询结果汇总后再返回给用户。

为获取最好的性能,消息通信服务的消息传递路由是可以配置的。通常,如果硬件支持,则选择广播;硬件支持不太好时,可以将消息直接传递到特定机器。

由于消息是基于 UDP 的,有时消息不能到达所有的机器,这就需要重新传递。消息通信服务会定期发出探测信息,记录当前所有活跃的存储机器数,如果应答数少于该数值,消息通信服务会在一定期限内重复传递消息。为了避免重复应答,每个消息按顺序赋予一个序号,并且包含了已经作出应答的机器列表。当存储机器接受一个消息时,它先检查自己是否已经包含在该列表里,如果没有,它才作出应答,同时,主存储机器会根据应答消息的序号和机器名,判断该消息是否重复应答。

#### 四、性能与优化

如上节所述,系统实现了 4 个主要的 API。下面阐述这些 API 的性能及所采取的优化措施。

根据 API 1) 和 2), 可以通过两种途径来访问网页:使用 PageID 访问或者使用 URL 来访问。对前者而言,它同时还存在两种方式:顺序读取模式和随机读取模式;对于后者而言,它是一种随机的读取模式。表 10-2 显示了这三种网页读取方式在一台普通机器上的性能。该表显示:使用 PageID 顺序访问方式的读取速度要远远大于其他两种方式,这也意味着高层应用程序应当尽可能使用这种访问方式。



表 10-2 网页访问性能(个/秒)

PageID 顺序访问	PageID 随机访问	使用 URL 访问
1200	185	55

对于 API 3),它的执行涉及所有的存储机器。不过,如果限制在单台机器上,一个 API 2)的执行实际隐含了一次 API 3)操作。这也意味着 API 3)的性能要比 API 2)的性能稍好一点。

对于 API 4),在访问机制上和 API 1)是相似的。不过由于 URL 和元数据比网页小很多,因此访问的性能相对好一点。

此外,使用多线程来处理请求可以提高随机访问的性能,这是因为磁盘访问的延迟(使用异步 I/O)和网络上传延迟可以被并行处理所缓解,同时,多个磁盘可以同时处理数据访问;在内存中设置缓存也有助于提高性能。例如,在很多情况下,最近被访问的网页(或者说 URL)很有可能会被再次访问。

## 第五节 网页的格式保存

Web InfoMall 的一个主要目标是要将持续搜集来的历史网页长期保存下去,使得后人可以再来回顾和研究这些网页。因此,需要制定一个适当的历史网页存储格式标准,使得机器可以正确地读取这些数据。

我们认为,原始网页的存储格式应当清晰明了,不依赖于特定的系统、硬件和软件,同时还应当易于理解和处理。此外,由于存储介质都是有寿命的,而且在数据存储和传输的过程中,也可能产生一些数据错误,所以应当考虑存储介质损坏时数据的可恢复性。例如磁盘的某个扇区损坏,导致部分数据不能读出,如果剩下的数据仍然可以使用,这就能将损失降到最低。

基于这样的考虑,提出一个原始网页的存储格式标准。参看第三章第二节的网页信息存储的天网格式。

在本格式标准制定之后,它也成为北京大学天网搜集系统的标准存储格式,这使得由天网搜集系统抓取的网页可以直接被 Web InfoMall 使用,给系统开发带来了便利。

## 第六节 小 结

在系统介绍国外历史网页保存现状的基础上,本章阐述了中国 Web 信息博物馆(Web InfoMall)的系统设计及实现的技术细节和策略,包括如何存储海量的历史网页,如何建立基本的索引数据,以及如何提供服务等问题。

从设计和实现过程可以看出,海量数据的处理是该系统所面临的最大挑战,这是

因为在数据量小时,很多问题容易解决,而当数据达到一定规模后,原有的方法就不再适用;而且这项工作工程性较强,意味着大量的程序开发和测试,因此需要良好的设计以避免走弯路。

现今,越来越多的国家和机构开展了 Web 信息保存的工作,也建设了一些 Web 仓储系统(Web Archive),数据规模最大的仍是美国的 Internet Archive,但该系统并没有公开技术细节,所以 Web InfoMall 系统的设计更多地参考了搜索引擎的相关存储系统,如 Google 等,它们也维护着一个非常大规模的网页存储系统,并且开发出了实用的大规模数据存储技术如 GFS(Ghemawat et al. 2003)、BigTable(Fay Chang 2006)等。

本章所实现的 Web InfoMall 仓储系统较好地解决了海量历史网页的组织、存储与长期保存等问题,具备较好的性能。目前,它已经存储了几十亿的历史网页,并且数据量还在持续增加之中。当然,Web InfoMall 还存在着很多不足,如网页数据的存储缺少冗余机制,数据在空间维度上的访问效率很低等。

本章内容取材于黄连恩同学的博士论文的第七章(黄连恩 2008)。

## 第十一章 大规模 Web 非网页信息仓储系统的构建

Web 上的信息由网页和非网页资源组成。上一章介绍了中国 Web 信息博物馆 (Web InfoMall 系统),它是一个网页信息的仓储系统。本章主要介绍由北京大学网络实验室开发的一个针对非网页信息的仓储系统——中国互联网数字资源财富库藏 (Chinese Digital Assets Library, CDAL),简称 CDAL 系统。

本章内容安排如下:第一节介绍网络资源库藏的相关工作,第二节是 CDAL 系统概况,第三节介绍系统设计方案及如何实现可扩展的存储组织;第四节介绍一种网络资源描述信息获取方法,该方法用于在互联网中为电影、书籍等热点门类资源寻找其相关描述文本;第五节介绍上述任务中使用到的一种词汇共现算法;最后是本章小结。

### 第一节 网络资源库藏相关工作

目前互联网上有很多专题资源收藏应用,比较著名的如互联电影库 IMDB(IM-DB 2008),软件库 SourceForge(SourceForge 2008)等;此外还有数字图书馆如亚历山大数字图书馆项目(Alexandria Digital Library Project)(ADL 2004);它们或者面向相对单一的资源类型,或者需要大量的人力维护。相比之下,我们更关注资源来自互联网用户的、综合性的、以低代价构建的海量库藏。这类库藏中比较著名的系统有 ibiblio(ibiblio 2008, Kahle 1997),Internet Archive(Archive 2012a)和 Wikimedia(Wikimedia 2008)等,本节将简要介绍这些库藏与本文所实现的系统 CDAL 在组织、存储、管理方式等方面的异同。

表 11-1 几个网络资源库藏系统的特征

	Ibiblio	Internet Archive	Wikimedia	CDAL
组织方式	简明分类体系和主题结合			
资源粒度	收藏对象的主题和内涵划分没有严格限制			
数据来源	用户提供	用户提供	用户提供	自动搜集+用户提供
存储方式	半集中	集中	集中	集中
管理方式	两步审核+ 用户自主维护	从数据到描述多步 严格审核,集中维护	完全依赖用户, 基于协作维护	对用户依赖少, 集中维护为主

这些库藏的共同之处在于:①资源种类多,大部分为互联网搜集或用户提供;

②高层类别数目少,层深浅;资源无法细分时会按照主题组织;③内容提供和内容审核。多数资源由用户提供,并附加描述信息,管理者负责审查质量并根据描述信息分类或加工元数据;④服务方式包括基于资源名的关键词检索、分类浏览及尽力而为的元数据检索;⑤资源的粒度不一,收藏对象的主题和内涵划分没有严格限制。这种自由性简化了人工组织的复杂度,但是给自动分类造成了难度。表 11-1 列举了这几个网络资源库藏系统的简单特征。

### 一、Ibiblio

Ibiblio 始于 1992 年北卡罗莱纳州大学建立的 SunSITE(<http://www.ibiblio.org>),收藏的资源包含多种 Internet 上可自由获取的收藏集,有免费软件、开源文档、图形、艺术、小说、诗歌、文学、音乐、宗教、政治、文化研究等多种内容,还提供流媒体内容访问。Ibiblio 在数字资源在线服务方面有悠久的历史,收藏量也很大,形成了全美最具历史的网上数字图书馆(Jones 2001)。平均每天访问量约 150 万。

Ibiblio 对网络资源的管理类似于开源社区管理开放源码的模式。特点是“贡献者驱动”(contributor-driven)和“内容自治”(content-managed)模式。它历来注重贡献者参与并鼓励自主管理,提供灵活的参与管理方式。

资源收集的方式:①向用户提供 telnet, ssh, ftp, scp 等连接方式登录 login. ibiblio. org 服务器,将有特色的收藏集收集到自己管理的 ftp,开放权限仍让原收藏人管理和维护内容;②提供虚拟主机,让用户建立或发布特色收藏;③以链接的方式,指向愿意公开共享收藏内容的站点。Ibiblio 收集资源的时候,要求用户提交描述表单,由人工或程序检查是否符合收藏准则。被认可后,由用户发布或上传到指定目录,再由人工或程序检查该内容之后才提供服务,这个周期最快要 1~2 天。Ibiblio 鼓励提供者把资源集的特色描述出来以便于检索。这些资源的元数据和组织结构都由内容提供者负责。

内容组织方式:Ibiblio 根据统一十进制分类法(UDC)提供分类体系,10 个大类,选用其中一百多个子类,共两层结构。每个子类中包括若干收藏集,每个收藏集有简短的文字描述说明主要内容。每个资源可能对应 ftp 上的目录,也可能是通过网页超链接组织起来的内容。

服务提供方式:分类树浏览收藏集和关键词检索。由于元数据依靠资源供应者建立,所以不是每个收藏集都提供专门的元数据检索。为了提高互操作能力,Ibiblio 的项目和服务提供元数据标准来保证信息交换,这些标准包括 Open Archive Initiative(OAI 2003),Dublin Core(DCMI 2008a)。除了浏览之外,用户可以通过 ftp 下载某些内容。

系统环境:硬件使用了 14 台 IBM、Dell 等服务器(它们提供的存储容量大约 5TB),分别用于主站点的 Web 服务及其冗余备份、软件、音频内容、镜像、捐赠上传、ftp 文件服务(占其中的 4 台)等功能。这些服务器都使用 Linux 操作系统,基于 Ar-

pache 提供 Web 服务。使用 MySQL, PostgreSQL 等数据库软件和 PHP 等开发软件。

## 二、Internet Archive

建立于 1996 年的 Internet Archive, 其使命是让全球的人都能够得到全部的知识与创作的作品, 并为研究者、历史学家和学者提供永久可获得的数字式历史收藏品 (Kahle 1997)。至 2008 年 1 月, Internet Archive 共收藏影视资料片 111 396 个, 声音类资料 223 768 段, 软件 33 516 个, 教学资料 1 276 个, 文字类资料 336 310 篇, 还有最为著名的 850 亿历史网页存档。收藏资源已达到 PB 量级, 每月还会新增 20TB 以上的内容。

资源收集的方式: Internet Archive 与 Ibiblio 不同, 几乎所有的数据都是集中存储在自己的设备上, 给用户提供免费、永久的存储空间, 供用户通过 ftp 等方式上传视频、图像、文字文件。这些资源经过 1~2 天检查周期上线。用户需要提供上传资源的标识符, 如果系统检测标识符唯一, 就在 ftp 服务器上建立一个目录, 用户必须在 24 小时内向这个目录上传或更改文件。之后系统自动检查目录, 并由内容管理员、系统管理员继续检查 (Archive 2008b)。

内容组织方式: Archive 按照分类和主题标引结合的方式组织资源。每个领域对应多个子集合, 这些子集合是根据资源主题来标识的, 其中又可以包含多个子主题。主题之间形成层次结构。这种方式非常适用于网络资源这种内容形式灵活的状况。

服务提供方式: 按照主题浏览及查询, 还提供元数据检索。存储在 Internet Archive 服务器中的文件可以用 http 或 ftp 的方式被其他人自由访问。用户可以对每个资源发表审阅评论。

资源存储管理: 每个被接受的资源都以目录的形式存放在文件系统中, 包括成员文件 (有些还包括适合不同传输带宽的版本)、XML 格式的元数据文件、其他人写的评论文件、资源的成员文件清单等。这种存储方式非常便于以资源为单位的转移、备份和再利用。存储设备使用 PC 机的 IDE 硬盘或 DLT 磁带, 选用什么格式存储取决于收藏集。

元数据: 不同种类的资源应用的元数据体系不同。以文本类为例, 采用 MARC 体系还是 Dublin Core 元数据集取决于收藏集来源, 但是每份同类资源都必有一个与这两者不是同一元数据体系的元数据文件, 这应当是 Internet Archive 为每类资源制订的自用标准。已有资源的元数据质量都很高。Archive 的资源质量高, 元数据信息较齐全。按内容提供来源不同, 有些元数据是由内容审阅者填写, 有些是资源提供者填写, 还有些资源直接链向提供元数据者。

## 三、Wikimedia

Wikimedia 最大的特色是引入宽松的用户参与机制建立资源库, 这是 Web2.0

应用的特色。创建和组织资源中,用户占主导,站点几乎不直接组织和管理内容,只起引导作用,制订参与规则和沟通的制度。在这种自由交互方式下,用户的积极性得到了极大的发挥,资源收藏量剧增,发展速度超过以往其他项目。

在这之前,自由共享 Internet 数字资源,并鼓励参与者贡献内容,或帮助组织内容的应用有很多,如上文提到的 Ibiblio。但是 Wiki 的编辑模式,使用户获得更大的参与自由,这使内容的贡献和组织更为活跃。

Wikimedia Commons(维基共享资源)是 Wikimedia 下的一个项目,对自由图片、音乐、音视频片断和文字、演讲访谈等内容提供集中存储,其中也包括一些有版权要求的资源。该项目 2004 年 9 月 7 日发布,第二年就收集到 20 010 个收藏集和 274 113 个多媒体文件。提供文件上传服务或特殊程序,其他用户可以用来上传图片或文件。

#### 四、中国互联网数字资源财富库藏

中国互联网数字资源财富库藏(Chinese Digital Assets Library, CDAL)系统是由北京大学网络实验室建立并维护的综合网络资源库藏系统。构建的目的主要有两个:一是保存网络社会精神财富;二是为海量中文数字资源的分类、存储、管理、描述、访问、检索等问题的研究提供平台,并为国内相关研究提供数据集。2003 年开始建立,至 2006 年 6 月已拥有 7.5TB 的存储容量,资源总量约 1.63 万,包括文件数 61 万。

资源收集方式:CDAL 的资源收集分为两个阶段:在 2003 年初到 2003 年 6 月的半年间,自动搜索中国互联网上公开共享的权威 ftp 站点获取资源;在此后的三年,提供 ftp 服务器,向所有访问者开放,收集网上资源。所有资源经过粗略的人工筛选,从原始目录树转移到 CDAL 分类体系中。我们没有雇佣专业人员进行精细的分类,而是在可接受的正确率情况下,利用资源原始组织知识,以目录树合并(目录归并)为思路,快速完成资源的初步组织。

内容组织方式:基于网络资源的特点,CDAL 以分类体系为主、专题收藏集为辅的内容组织策略。分类体系共有 7 个领域,对应 7 棵分类树,分别是声音(内容为声音类的资源)、影像(内容为静态或动态的图形图像类资源)、文字(内容为文字类的资源)、软件、交互式资源、事件和数据集。每棵分类树对应 2~5 层的深度,共有 455 个叶子类别。此外,还有专门针对网络资源集合的“成套收藏”和专题收藏。

传统文献的分类法(如中图法、美国国会图书馆分类法(LCC 2008)、统一十进制分类法(UDC 2006)、杜威十进制分类法(DDC 2008))追求理论严谨、体系科学、分类结构和规则完整,提供并使用类目受控词汇表。而网络资源信息量庞大、内容形式灵活多变,传统分类体系不易适应。目前为止还没有针对网络资源形成统一的、权威的分类法。构建合理实用的网络信息资源分类法,是建立资源库藏值得仔细考虑的问题之一。

CDAL 分类体系中没有采用 LCC、UDC 等分类法,一方面是因为这些分类法侧

重于为书籍等传统媒体分类服务,另一方面是因为它们是基于西方文化思想进行的知识内容组织方式,而 CDAL 中多数库藏是以中文内容为主的网络资源。

服务提供方式:CDAL 提供基于名字(目录名、文件名)的关键字检索、专题和分类浏览方式,并提供热点资源的相关文字描述。这些文字描述并非人工整理,而是利用热点资源在互联网上的网页信息自动抽取的。

资源存储管理:每个资源都以目录的形式存放在文件系统中,包括成员文件和元数据模板文件。这种存储方式非常便于以资源为单位的转移和再利用。与 Internet Archive 的不同之处在于,CDAL 将分类体系映射为文件系统目录树,每个磁盘具有同样的目录结构,并与总体分类体系一致。这样的结构借鉴了“分治”思想,每块磁盘都可被当作整个系统的缩影,提供资源存储和访问服务,因而该结构具有良好的可扩展能力。

CDAL 系统用树状存储组织模式管理资源。Ibiblio、Internet Archive 等系统的资源收集都依赖于用户,而且要求用户整理资源内容并填写元数据等信息。这种做法实际上是把繁重的组织内容工作从管理端转移到用户端,因而对用户的要求较高,不适用于一般的希望共享而又不愿意做太多额外工作的惰性用户。我们所构建的 CDAL 系统与此不同。因为惰性用户在互联网上占绝大多数,收藏机制能够降低共享门槛无疑是受欢迎的。这是我们独特的着眼点,也是驱动我们研究自动收集、组织和处理网络资源关键技术的重要原因。

## 第二节 CDAL 系统概况

CDAL 的工作始于对中国历史网页保存(InfoMall 2002)的启发,因为不同形式的网络资源都是人类在互联网社会中精神财富的体现,经历着网络环境下的创建、发布、共享传播和消亡的过程,不像传统媒体信息那样有专门的机构发行、存档,具有产生快、易于流逝的特点。目前 CDAL 的资源无偿共享给 P2P 存储共享系统 AmazingStore(<http://www.amazingstore.org>),该系统中“7T 经典资源,保留记忆中最难忘怀的片段”就是指 CDAL 资源。

开展这项工作的目的有两个:①收藏,不仅包括热门类别网络资源,还注重用户自己创建、组织的内容。②研究,关注网络资源从无序的原始状态到达有序组织过程中,有哪些环节可以用何种省力、省事的方式去完成。

CDAL 系统构建包括资源收集、组织分类的策略和方法,存储体系的设计与实现、系统管理、服务等方式。除了保存网络资源历史,本项工作的意义还在于:探索了低成本、快速灵活、可扩展地构建适于海量网络资源存储组织管理系统的模式。

CDAL 提供的服务主要有 3 种方式(图 11-1):基于分类体系的浏览(分类树见左侧框)、基于名字关键词的检索(检索结果见中间框)和基于专题的收藏(见右侧框)。

由于每个资源有自身的内部构成,一些资源是某种作品集,可能包括关于一个事件的多个图片,一个作者的多部作品,或者一个专辑的多首歌曲,所以 7.5TB 的 1.63 万资源事实上包括了更多的单个实体。这里选出 13 个常见类别,其资源量和字节数分布如表 11-2 所示。



图 11-1 CDAL 提供的资源访问方式

表 11-2 CDAL 中的资源分布

类别	资源数比例/%	类别	资源数比例/%
书籍	25.9	乐曲	3.80
文章	3.26	图片(含照片绘画)	3.94
电影	18.9	讲义	1.94
电视	8.97	听力	0.72
软件	9.70	视频资源	0.37
歌曲	18.9	游戏	3.14
戏剧曲艺	0.46		

考虑到网络资源的门类综合,题材多样,形式自由,数量庞大等因素,CDAL 分类体系建立时遵循如下原则:



1) 易读性,层深 2~5 层,每个类别含义明确。

2) 兼容性,各大类别参照现有领域标准或惯例综合而成,如“书籍”参照中国图书分类法;“电影”参考 IMDB 等大型影视网站的分类和学院派观点。

3) 灵活划分,主要是与资源题材和组织风格相适应。CDAL 分类体系首层类别名选自 Dublin Core 元数据集中关于资源类型(Type)的推荐词汇<sup>①</sup>(DCMI 2008c),如“声音”、“影像”、“文字”等。第 2 层是上述类别下所列的样例词汇,如声音类包括“歌曲”、“乐曲”等,文字类包括“书籍”、“文章”等;在更细致的层次上,结合网络资源的特点和惯例,还按照内容的题材、表现形式、风格流派、人物、地域等属性划分。

由于在 CDAL 建设初期,资源组织由人工完成,而分类体系过于复杂精细会对资源组织人员造成过重的负担,因此领域知识划分主要采用了便于利用、简明通俗的基本原则。实践表明整个分类体系基本能够覆盖网络资源的内容范畴。

### 第三节 CDAL 系统设计

#### 一、系统体系结构

系统设计原则为:系统的可扩展性,可用性,易维护。采用图 11-2 所示的分布式系统结构,有多台服务器,几乎每台服务器的部署都相同,既承担存储又提供访问服务,存储的内容各不相同,但是支持访问的索引是一致的,全局索引记录了资源所在的实际位置。来自互联网的请求首先被系统前端的负载分配器导向到某服务器  $i$ ,  $i$  检查索引表,如果资源实际存放在本地,就直接提供服务,否则将请求递交到资源所在主机为用户传送数据。

按照模块化设计思想,定义各服务器的功能几乎是相同的,而每个磁盘的角色也一样。这种设计在存储、管理资源的能力上有线性增减的优点,具有很好的可扩展性。而且一块磁盘或者一个节点的失效,只是局部不能访问,不会影响其他磁盘或者节点的服务。系统的任何一部分独立出来都能很容易构成一个相似系统,相应地,当资源量增长的时候,也能够适应数据规模的增长。

#### 二、可扩展的存储组织方案

CDAL 以树状方式刻画资源的构成,并将分类体系的知识组织映射为文件系统的存储组织,且每块磁盘都保持一致的目录结构。这种模式使资源在进行再组织时,不同目录树之间迁移的时间  $t_m$  几乎为 0。

设计资源组织任务的存储方案,主要考虑以下几点。

---

<sup>①</sup> 该词汇集 2000 年提出,历经多个版本的修改,最近的版本是 2008 年 1 月更新的,已从原来的 8 个大类别细化为 12 个大类别。

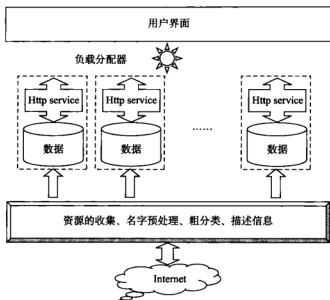


图 11-2 CDAL 系统结构图

1) 原始组织形式。原始状态的资源也是存在于文件系统中,并有自己的内容和结构,其成员可能通过多层子目录组织。

2) 存储粒度。资源和文件的大小都不定,不适合结构化数据管理方式。

3) 访问方式。库藏系统中的资源一旦被存储,很少需要修改,多是读取操作。

4) 迁移。出于均衡负载的考虑,以及使设备升级而引起部分数据存储位置发生变更时,要减少对服务的影响,并用简便的方式完成。

CDAL 选择文件系统存储,这样做的好处显而易见。例如,文件系统是计算机管理最成熟的设备之一,资源容易备份、复制和转移,也容易远程操作,不需要额外安装管理软件,系统升级维护都简单方便等。

当然,最大好处是允许将资源组织体系映射到存储体系上。将分类树以文件目录树的形式反映出来,并复制到存储系统的每个磁盘上。这种设计对资源的初期组织、迁移、备份、整合都有极大便利,重构和维护的成本都很低廉。下面详细阐述其优点。

1) 目录名即类别名。CDAL 建设初期,分类以目录合并的方式完成。从互联网 ftp 上批量收集的资源是带有原始组织结构的,被存储在一块块磁盘上等待分类。多人处理多块磁盘中存储的资源,将其从原始目录树  $S$  转移到 CDAL 分类体系映射的目录树  $D$  中。由于每块磁盘都有同样的目录树  $D$ ,能用来组织各类别的资源。只涉及同一磁盘中 move 操作,而不需要将资源在磁盘之间复制。这种操作就是修改操作系统文件节点表等信息,不发生文件物理存储位置变化,所以时间开销非常小。

2) 磁盘级的相似性。如果将不同类别的资源存储在不同的磁盘上,有些热门类别的资源必将面临较多的访问,这样磁盘负载不均衡,导致频繁读的磁盘易损坏。现在这个方案就不存在这个问题,同一块磁盘上可以有各种类别的资源,分担读盘压力。

3) 管理和维护成本低。资源备份时,既能通过整盘复制保存磁盘所有资源,又能按类别复制,即指定每块磁盘特定类别目录的复制。迁移也是同样。只要有一块磁盘在一台服务器上工作,系统都不会终止响应。虽然位于损坏磁盘上的资源无法下载,但是全局索引还可以使系统应对浏览和查询。

使用文件系统也有不便之处,主要是磁盘空间的使用不经济。例如:

1) 缺少好的内容复用机制。当一个文件可以属于多个资源时,虽然文件系统提供了软链接机制,但是资源位置转移会引发链接失效。最直接的办法就是允许文件的多个复制,但这样浪费存储空间。

2) 资源的文件大小不一,而磁盘是分块顺序存储的,如果不使用专门的存储碎片压缩机制,可能会造成磁盘空间的浪费。

但是考虑到硬件存储设备价格下降、容量增大的趋势,可以少量的局部冗余和磁盘空间浪费,换取了便捷高效的管理和维护。

## 第四节 网络资源描述信息获取

元数据、摘要都可以起到资源描述的作用。当需要满足的查找目的不同时,使用的方式也不同。元数据描述的目的是为了检索,通常涉及对象的多种属性,描述质量较高,支持结构化精确查询,但是这种描述方式是耗时而昂贵的;摘要则在信息浏览中起重要作用,搜索引擎对网页的摘要就是一个典型的应用例子,用户不必一一查看网页,通过摘要就能够决定该网页是否为所需要的内容。

考虑到多数用户访问或下载资源之前,更想知道的是对内容的描述及来龙去脉,相关文字描述可以帮助用户判断是否为所需内容。而且现阶段,摘要形式的资源相关描述文本比元数据更现实。

首先,精确元数据的获取成本较高。每个网络资源都是复杂数据集合,它们通常单体传播,缺少上下文,且本身很少携带文字介绍。没有通过专门训练的人即使对内容已经了解,也不易给出恰当的描述;而激励用户贡献的做法又往往因为理性用户<sup>①</sup>的存在而收效欠佳。

其次,网络资源的内容不够规范。数字图书馆领域提出元数据收割协议 Open Archives Initiative-Protocol for Metadata Harvesting(OAI-PMH)来帮助成员馆之间元数据共享、减轻著录负担(OAI 2003)。但是多数网络资源并没经过规范的内

<sup>①</sup> 理性用户指贡献少、索取多的参与者。

容组织,而是网络用户自由创建和发布,难以找到它们的“收割源”。

本节主要关注热点门类<sup>①</sup>的资源,提出一种基于 Ontology 的概念扩展方法,从搜索引擎返回的网页中提取网络资源描述信息。借鉴 Dublin Core 核心元数据对实体的描述,依据分类体系构建领域 Ontology,通过等价类词汇扩展和属性词汇的扩展,对搜索引擎结果定向过滤,既避免了返回过多内容,又能够保证过滤出的信息相关且全面。描述信息消除了网页噪音、相关程度较高、文本量少。

2004 年 CDAL 系统使用这种方法为 5 个领域的近万种资源提供描述信息。本节主要介绍设计、实现及对描述信息排序过滤的方法。

### 一、Ontology 概述

Ontology 是面向领域的,是领域概念模型的描述,用公认的术语集和术语间的关系来反映该领域内的知识和知识结构;有概念、属性、公理、取值和名义等构成要素,其中属性是对概念特征或性质的描述(邓志鸿 et al. 2002,宋炜等 2004)。

体系分类法是一种典型的 Ontology,分类体系主要依据概念划分与概括原理,将绝大多数资源内容及形式特征的主题概念列举为类目,通过概念的范畴划分为类别等级体系。体系分类法能较全面和客观地反映知识体系的全貌及内在逻辑联系。每个类别在分类体系上都有相对固定的位置和次序,相对比较稳定。多数类别的上下层次之间构成一种 IsA 关系。

### 二、描述信息获取机制

分类体系是 Ontology 的一种反映,类别对应概念,概念有若干属性,这些属性可以选用 Dublin Core 元数据项来表述。目的就是从已有的 CDAL 分类体系出发,结合主要的元数据项,利用搜索引擎获取高相关度的资源信息。具体方案如图 11-3 所示。

提交搜索引擎查询的基本关键词集  $Q$  取自资源名  $R_n$  和分类体系的类别名  $C_i$ 。对前者的处理包括切分出语义信息,选取名称、作者等实体名语义片段充当查询词。对后者的处理包括建立等价词汇集合  $W_{C_i}$ 。因为分类体系的类别  $C_i$  使用规范或受控词汇命名,和网络中多数内容活泼自由的用语风格不匹配,要进行适当同义或近义转换,目的是为了贴近查询。每个资源实体对应一个词汇集  $Q$ ,尝试查询串  $\forall q \in Q$ ,  $Q = \langle R_n \times W_{C_i} \rangle$ ,得到相关度排在前  $k$  的网页。所有  $Q$  中查询串对应的结果集  $R$  作为获取描述信息的“参考信息源”。最后提取摘要生成网络资源的描述信息  $D$ 。

每次查询中都要选高相关度网页,改进原始排序的策略是:基于类别概念对应的属性构造扩展词汇集  $A$ ,通过特定的排序规则突出相关度高的网页集合  $R$ 。

<sup>①</sup> 人们认为那些公开发行的资源为热点内容,如影像、软件、书籍、音乐等,互联网网页中通常会有这类资源的相关描述。

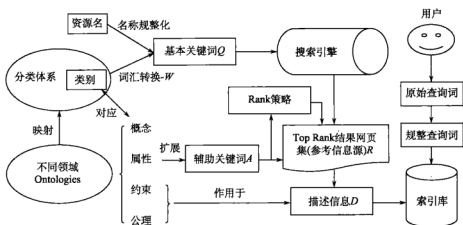


图 11-3 基于 Ontology 的网络资源描述信息获取

由于  $R$  中网页来自同一个资源实体信息构造的多组查询,可能有重复网页,对网页链接和内容均作 md5 检查,消除这种情况,则  $R' = \bigcup \{R_{q_i} | q_i \in Q\}$ ,  $|R'| \leq k|Q|$ 。

需要说明的是 Ontology 具有静态性和动态性,除了刻画概念模型之外,还可以通过定义公理、规则辅助进行概念的推理,本节所述的工作并未进行推理这一环节。

这个过程中,使用北京大学网络实验室的天网搜索引擎检索相关网页,使用了天网网页净化(张志刚 2004)工具去除页面噪音,来保证相关网页的质量。

### 三、改进查询的方法

为了丰富查询词,对叶子类别  $C_i$  的类名衍生出一个等价词集  $W_{C_i}$ 。目的是为了将分类体系中严谨而学术性的名称用相同或相似的词汇表述,例如类别“诗歌韵文”转换为“文学”或“诗”。具体做法包括:从各类别中选取典型资源做种子,以种子资源名字做关键词搜索互联网上已有的专业资源库;找到该资源在不同资源库分类体系下的类别标签。这种方法适用于电影、歌曲、书籍等热门类别。

资源的原始目录结构或资源原始名称中也可能带有类别信息。这种包含在原始命名和组织结构中的信息也可以利用。

取  $|W_{C_i}|$  为  $1 \sim 2$ ,  $k \leq 8$ , 每个资源  $r$  的基本信息源  $R_r$  所含平均网页数约为 13, 即每个资源平均有 13 条相关网页,构成用于筛选描述信息的参考信息源。网页的题材有新闻、网络文学、数据库中的信息等。还需要用下一节的方法从  $R_r$  中过滤出所要的描述信息条目,对其相关性重新排序。

### 四、改进排序的方法

改进排序的理由是:2004~2005 年开展本工作时,使用天网搜索引擎直接返回的检索结果并不是非常有效;通用搜索引擎对结果排序一般要考虑更多因素,而对特

定目的的检索需求,使用策略调整返回结果是一种自然而然的做法。用类别的属性词及共现词调整搜索引擎返回的检索结果排序。

对不同类别,从 Dublin Core 元数据集选出适于作为类别属性的项,并加以词汇扩展,目的是能从参考信息源中得到更相关的描述信息。这样,每个类别对应一个属性集  $A$ ,这些属性名词提供辅助关键词。定义  $A$  中各元素权值序列。以“影像”类的 Ontology 为例阐述上述过程,如图 11-4 所示。

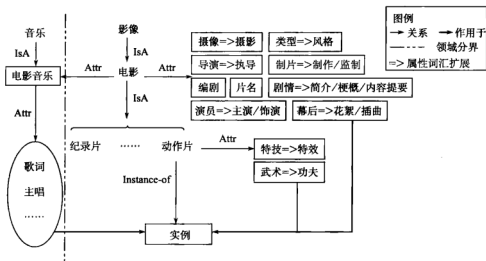


图 11-4 概念的属性及其词汇扩展(以电影类资源为例)

概念的属性及词汇扩展表达了丰富的内涵,概念之间为 IsA 关系时,下层可以继续上层概念的属性。一个影片实例属于动作片,由于父类概念“电影”及“动作片”的属性可“传递”到实例上,在筛选网页时,除可用资源实例名和“动作片”这样的类别名查找参考信息源外,还可进一步采用属性扩展构成的辅助关键词找到与资源关联最紧密的描述信息。属性词汇扩展的目的是为了用更普通常见的词替代元数据项那种规范的学术化名称。这个过程和类别等价词的获取相似,但目的不是为了丰富查询,而是改善排序相关度,也不只使用等价词汇,而用到了更多信息,如共现词汇。在下一节将介绍一种基于局部聚类思想的词汇共现算法 FDC。

用资源名中的有用信息(如经过语义片段切分得到的主要片名)、属性集  $A$  中元素和对应权值过滤返回参考信息源  $R$ ,形成描述信息集  $D$ ,并影响对描述信息条目的排序,同时生成句子摘要。

每个资源对应的参考信息源要用辅助关键词帮助过滤和排序。不同辅助关键词在表达概念相关性上有区别,如对一部影片而言,表达影片内容的词汇权重应当高于表达影片音乐的作曲风格这样的词汇。算法如图 11-5 所示。

输入：资源  $r$  对应的参考信息源  $R_r$ ， $r$  所属类别  $C_i$  的属性词集  $A_{C_i}$ ，及  $A_{C_i}$  中元素权值序列

输出：过滤并排序后的  $R_r$  的子集  $R_r^1$

- 1: 获取  $R_r$  中每个网页文本
- 2: 在文本中依次查找  $A_{C_i}$  每个元素的出现情况，记录在以网页号为行，辅助词为列标记的矩阵中。同时提出辅助关键词在文中出现位置的上下文作为摘要
- 3: 上一步中形成的矩阵与  $A_{C_i}$  权值序列的转置阵相乘，获得  $R_r$  中每个网页的权重
- 4: 对 3 排序，保留权值在前  $n$  位的网页为资源  $r$  的描述信息
- 5: 输出描述信息集合  $R_r^1$ ，及其中每个网页的摘要

图 11-5 获得描述信息的改进排序算法

上述算法得到的结果只是机器认定的最优，向用户展示描述信息的时候，还要提供人为选择相关性的交互式渠道，记录用户选择结果，并滤去那些没有或不经常被选择的网页，如图 11-6 所示。



图 11-6 网络资源描述信息显示

## 第五节 基于局部聚类思想的共现词汇算法

共现词汇在信息检索的排序和查询扩展中都有较多应用，同时共现词典还是自然语言处理中的一种工具。利用局部聚类思想，本节提出了一种新的共现词汇算法，称之为 FDC 算法。该算法主要考虑了词汇在文档中的共现频度 (co-Occurrence Frequency)、相对距离 (Distance) 和共文档率 (co-Collection Ratio) 等三个因素。

共现 (co-Occurrence) 是指词汇在文档集中共同出现。以一个词为中心，可以找到一组经常与之搭配出现的词，称为它的共现词汇集，这个集合描述了该词的语义上下文特征或语境。

文献 (Attar et al. 1977) 提出关联聚类和距离聚类均有助于发现文档中的共现词汇。文献 (Beefenman et al. 1997) 的研究也表明词汇的距离与共现紧密程度有关。除此之外，潜在语义索引 (LSI) 也可以用于发现共现词汇 (Berry et al. 1995)。参照上述研究本文提出一种新的共现词汇算法 FDC。

## 一、基本定义

对检索系统,任给关键词  $k$ ,从返回的结果文档  $D(k)$ 中得到的词汇集合  $T(k)$ 。其中  $|D(k)|=n, d_j \in D(k), j \in [1, n]; |T(k)|=m, t_i \in T(k), i \in [1, m]$ 。通过某种函数  $A$  得到  $k$  的共现词集合  $S_k \subset T(k)$ 。函数  $A$  包括如下三个方面的因素。

1) 词频:  $\forall t_i \in T(k)$  且  $t_i \neq k, t_i$  在  $d_j$  中出现的词频为  $f_{id_j}$ , 如果不出现则  $f_{id_j} = 0$ 。 $t_i$  与  $k$  在  $d_j$  中共现频度为  $f_{id_j} \cdot f_{kd_j}$ 。

2) 距离:  $t_i$  与  $k$  在  $d_j$  中距离远近也反映了两共同出现时的亲密关系,词间距离指两词间词汇的个数。因为在文档  $d_j$  中两个词均可能多次出现,取它们在共现时的最短距离,用  $r_{(i-k)d_j}$  表示。如果  $t_i, k$  没有共同出现于  $d_j$ , 则  $r_{(i-k)d_j} = \infty$ , 实际计算中将其设为构成  $d_j$  的所有词汇数。

3)  $t_i$  相对于  $k$  的共文档率: 在  $k$  出现的文档集  $D(k)$  中同时出现  $t_i$  的文档数  $|D(k) \cap D(t_i)|$  与  $|D(k)|$  的比。

其中,词汇的共现频度和距离,这两个因素体现了关键词及共现词在同一文档中的关系,共文档率体现的是关键词及共现词在整个文档集中的关系。

## 二、FDC 共现词汇算法

用  $F_{i-k}$  表示  $t_i$  与  $k$  在文档中的共现频度关系;  $R_{i-k}$  表示  $t_i$  与  $k$  在文档中的距离关系;  $P_{i-k}$  表示  $t_i$  与  $k$  共文档比率。

$$F_{i-k} = \sum_{d_j \in D_k, j=1}^n (f_{id_j} \cdot f_{kd_j}) \quad (11-1)$$

$$R_{i-k} = n / \sum_{d_j \in D_k, j=1}^n (1/r_{(i-k)d_j}) \quad (11-2)$$

$$P_{i-k} = (|D_k \cap D_{t_i}|) / |D_k| \quad (11-3)$$

由此定义  $C_{i-k}$  表示在包含关键词  $k$  的所有文档中,  $t_i$  与  $k$  共现的密切程度如式 (11-4) 所示。

$$C_{i-k} = \frac{F_{i-k} \cdot P_{i-k}}{R_{i-k}} \quad (11-4)$$

对  $\forall k_l \in K, l \in [1, L]$ , 利用式 (11-4) (实际计算中做归一化), 可以求出反映  $T$  中每个词与  $k_l$  共同出现情况的一系列  $C_{i-k_l}, i \in [1, m]$ 。排序取前  $u$  个元素, 找到  $T$  中对应的词, 就构成按照 FDC 方法所求的  $k_l$  的共现词汇集  $S_{k_l}$ 。

对于中文和英文的情况, 上述公式的细节处理有所不同。如果是中文, 文档需要事先切分成词, 词汇距离才可以求出。如果是英文, 比较不同词的最短距离时, 要转换大小写以便确认两个不同位置的词是否相同。

本算法曾用于制作搜索引擎用户查询词的共现词典, 目的是帮助改进天网搜索引擎排序。结果证明其有助于提高相关网页排名(陈翀等 2005)。



## 第六节 小 结

CDAL 系统是国内首个用于收藏组织综合门类网络资源的平台,设计思想的基本出发点是用尽量少的人工开销、适应海量资源的增量高效存储、组织。在初期阶段,资源组织依靠人工,正是这样,才引发了我们思考节省人力、缩短时间的技术、工具和方法。

系统的设计运用了模块化设计思想,每块磁盘和服务器功能部署相似,任何一部分都可以独立组合。整个系统具有较高可扩展性,能适应增量存储、组织和服务需求。数据存储组织采用树状结构,将分类体系对资源的逻辑组织和文件系统对资源的物理存放统一起来。这种方案的最大优点是降低了管理和维护的成本,并提高了资源组织效率。

作为一种访问服务方式,结合分类体系所体现的概念知识,提出并实现一种基于 Ontology 的资源描述信息获取方式。对热点门类的网络资源,结合类别概念扩展、属性词汇扩展、通过搜索引擎返回结果提取相关度高的简短摘要,改善了资源的访问。这种基于概念及属性扩展的具体做法还可以应用于主题搜索领域的信息获取中。

本章内容取材于陈肿同学的博士学位论文第七章(陈肿 2008)。

## 第十二章 中文网页自动分类与聚类

为了能够有效地组织和分析海量的 Web 信息,人们希望能够按照其内容实现对网页的自动分类与聚类。目前,网页自动分类与聚类技术在数字图书馆、主题搜索、个性化信息检索、搜索引擎的目录导航服务、信息过滤、主动信息推送服务等领域得到了广泛地应用。

本章内容安排如下:第一节介绍了文本自动分类算法的类型与特点;第二节给出了实现中文网页分类的一般过程。第三节首先定量地分析了影响分类器性能的各种关键因素,然后根据实际的测试结果并结合搜索引擎这一特定的应用环境,寻找一种中文网页分类器的最佳设计方案;在具有较高分类质量的同时,还具有较高的分类效率。最后,根据这个方案实现了一个能够处理海量中文网页信息的分类器。第四节介绍了如何应用中文网页自动分类方法实现搜索引擎目录导航服务(冯是聪 2003)。第五节简单介绍了文本聚类的主要方法。

### 第一节 文档自动分类算法的类型

在 Web 出现之前,人们已研究过许多普通文档分类的方法,形成了各种文档自动分类(Automatic Text Categorization, ATC)技术(Yang et al. 1999)。随着海量网页信息的涌现,ATC 技术的处理对象从普通文档扩展到网页信息,自然地,ATC 技术成了实现网页自动分类的基础。所谓文档自动分类,就是指用计算机程序来确定指定文档和预先定义类别之间的隶属关系(Sebastiani 1999)。

目前,主要的文档自动分类算法可以分为三类。

1) 词匹配法。词匹配法又可以分为简单词匹配法和基于同义词的词匹配法两种。简单词匹配法是最简单、最直观的文档分类算法,它根据文档和类名中共同出现的词决定文档属于哪些类。很显然,这种算法的分类规则过于简单,分类效果也很差。基于同义词的词匹配法是对简单词匹配法的改进,它先定义一张同义词表,然后根据文档和类名以及类的描述中共同出现的词(含同义词)决定文档属于哪些类。这种分类算法扩大了词的匹配范围,在性能上要优于简单词匹配法。不过,这种算法的分类规则仍然很机械,而且同义词表的构成是静态的,对文档的上下文不敏感,无法正确处理文档中其具体含义依赖于上下文的词,分类的准确度也很低。

2) 基于知识工程的方法。基于知识工程的文档分类方法,需要知识工程师手工地编制大量的推理规则,这些规则通常面向具体的领域,当处理不同领域的分类问题时,需要不同领域的专家制定不同的推理规则,而分类质量严重依赖于推理规则的质

量。因此,在实际的分类系统中较少使用基于知识工程的学习法。

3) 统计学习法。统计学习法和词匹配法在分类机制上有着本质的不同。它的基本思路是先搜集一些与待分类文档同处一个领域的文档作为训练集,并由专家进行人工分类,保证分类的准确性,然后分析这些已经分好类的文档,从中挖掘关键词和类之间的联系,最后再利用这些学到的知识对文档分类,而不是机械地按词进行匹配。因此,这种方法通常忽略文档的语言学结构,而用关键词来表示文档,通过有指导的机器学习来训练分类器,最后利用训练过的分类器来对待分类的文档进行分类。这种基于统计的经验学习法由于具有较好的理论基础、简单的实现机制,以及较好的文档分类质量等优点,目前实用的分类系统基本上都是采用这种分类方法。

本章介绍的文档分类算法都属于统计学习法。根据分类结果的不同,基于统计学习法的分类系统在整体上可以被分为两类:独立二元(Independent Binary)分类系统和  $m$  元(M-ary)分类系统(黄菁萱等 1998)。所谓独立二元分类,就是给定一篇文档,分类系统对每一个类都独立地判断这篇文档是否属于该类;要么属于,要么不属于,而不存在其他的结果,并且在分类过程中,不同类别之间互不影响。所谓  $m$  元分类就是给定一篇文档,系统计算这篇文档与所有预先定义的类的相似度,并按这篇文档和各个候选类的相似度排序,最后输出候选类列表。文档分类算法如图 12-1 所示,在第三节介绍其中几个典型的分类算法。

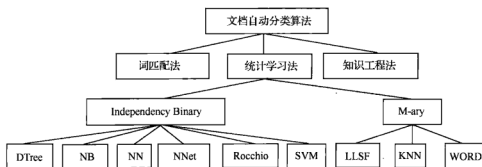


图 12-1 自动文档分类算法的分类

## 第二节 实现中文网页自动分类的一般过程

在应用基于案例的有指导的机器学习方法实现中文网页自动分类的过程中有一个基本的假设:文档的内容与其中所包含的词有着必然的联系,同一类的文档之间总存在多个共同的词,而不同类的文档所包含的词之间差异很大。因此,分类器的训练过程可以看做是在已知文档类别的情况下,统计不同类别内的词的分布,即在预先定义类别集合  $C(C=\{c_1, \dots, c_k, \dots, c_m\})$  与词项集合  $T(T=\{t_1, \dots, t_k, \dots, t_n\})$  的幂集

之间建立一种加权的映射关系,形成一种向量表示;相应的,分类器的分类过程,可以看做在已知一篇文档内所包含词项分布(用一个向量表示)的情况下,和在训练中形成的每个类别的向量表示进行对比,来确定该文档与类别的隶属关系。

根据对文档分类过程实质的分析,下面给出中文网页自动分类的一般过程。同普通英文文档相比,中文网页信息具有特性:① 中文网页的内容使用中文书写,不像英文单词之间存在自然的形态间隔,中文需要分词处理。而且分词的效果能够显著地影响分类效果;② 网页使用超文本设计。它包含大量的 HTML 标签和超链接。我们有可能利用这些信息来改进分类的质量。如包含在标题<title>标签内的内容通常要比出现在网页正文<body>标签内的内容要重要得多。在 Web 上相邻的网页通常具有相关或相同的主题,因此网页之间的超链信息也可以给我们一些启发;③ 网页通常包含大量的“噪声”。同普通文本相比,网页的设计比较随意,通常包含各类广告,设计人员的注释以及版权申明等无关信息。有时同一个网页甚至会包含多个不同的主题。在进行分类之前,需要自动清除这些“噪声”,否则这些“噪声”会降低分类质量。因此,需要对中文网页进行预处理后,才能应用相应的文档自动分类算法实现分类。

结合中文网页的特性,图 12-2 给出了中文网页自动分类的一般过程。其中:预处理过程主要包括中文分词以及网页内“噪声”的清除等处理;基于二元分类算法的分类器,可以把分类结果直接作为待分类网页的类别结果,而基于  $M$  元分类算法的分类器,还需要对该分类结果进行进一步的筛选后,才能作为待分类网页的类别结果。

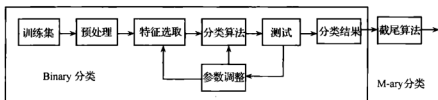


图 12-2 中文网页自动分类的一般过程

根据图 12-2 所示的中文网页分类的一般过程,我们设计了本章研究所使用的分类器,其工作原理如图 12-3 所示。从总体上,分类器的整个工作周期可以分成训练过程和分类过程。在训练过程中,训练集实例经过中文分词和特征选取处理后被表示成向量形式。该特征向量集用来描述类别模式,在分类过程中使用。校验集是训练集的一部分,通过应用相应的阈值策略来预先确定每个类别的截尾阈值。在分类过程中,一个待分类的中文网页经过中文分词并表示成向量后,应用分类算法同训练过程得到的类别模式逐一比较,得到候选类别列表,然后同训练过程中得到的每个类别的阈值相比较,保留大于阈值的类别,并作为该网页的分类结果。

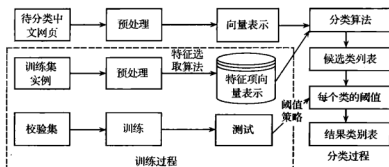


图 12-3 中文网页分类器的工作原理图

从图 12-3 可以看出,构建一个分类器的关键因素包括:预处理、训练集、特征选取算法、分类算法和截尾算法等。预处理部分的 HTML 网页净化方法已在第七章中介绍,如下将逐一地分析后 4 个因素对分类器性能的影响。

### 第三节 影响分类器性能的关键因素分析

在信息检索领域,评价一个系统的性能,通常有效果和效率两个方面的考虑。与此对应,评价一个分类器性能的优劣,通常也有两个基本的指标:分类质量(效果)和分类效率。对于分类质量,考察的指标通常为查准率和查全率;对于分类效率,考察的指标通常为分类器的训练效率和分类器的实际分类效率。分类质量和分类效率这两个指标,既相互独立,又相互影响。在理想的情况下,人们追求分类器不但要具有较高的分类质量,而且还要具有较高的分类效率。但是在实际的应用中,有时为了追求分类质量而不得不牺牲分类效率,有时为了保证一定的分类效率而不得不在一定范围内牺牲分类质量。因此,在设计分类器时,需要根据具体的应用环境来综合考虑这两个指标,重点解决主要矛盾。

#### 一、实验设置

为了定量地分析影响分类器性能的关键因素,首先实现了一个最基本的中文网页分类器。该分类器的具体设计方案如下:

1) 预处理。在预处理阶段,除了进行中文分词处理外,没有进行其他任何预处理。

2) 特征选取。在这里,直接把中文分词得到的所有关键词作为特征项,并由这些特征项构成特征向量,因此没有特征选取处理过程。

3) 分类算法。选用 kNN(k-Nearest Neighbor)分类算法来实现基本的分类器。在实验中取  $k=20$ ,即仅保留相似度最大的 20 个实例网页。为确定待分类网页的类

别,首先需要把具有相同类别的实例与待分类网页之间的相似度相加作为待分类网页的类别相似度,最后把相似度最高的类别作为该网页的结果类别,所以这里每个待分类网页仅取一个结果类别。

4) 截尾算法。因为上面的分类算法为每个待分类网页仅取一个结果类别,所以这里无需对分类结果应用截尾算法。

5) 分类质量的评价指标。在信息检索领域,通常采用查准率和查全率,人们通常借鉴这些标准来评价分类系统的优劣。

查准率表示在所有被检索出的文档结果集中,真正符合检索意图的文档所占的比率,它体现了系统检索结果的准确程度。查全率表示被检索出的文档集结果中真正符合检索意图的文档数在所有符合检索意图的文档集中所占的比率,它体现了系统检索所有相关文档的完备性。查准率和查全率这两个标准是互补的,单纯提高查准率就会导致查全率的降低,反之亦然。因此,尽管一个好的检索系统应该同时具有较高的查准率和较高的查全率,但是实际的检索系统往往需要在两者之间做出一些折中,而避免其中一个指标过低。

为方便起见,人们还定义了一个  $F_1$  值(Yang et al. 1999),用以反映查准率和查全率的综合效果,其定义如公式(12-1)所示。根据计算方式的不同, $F_1$  值可以分为宏观  $F_1$  值(Macro- $F_1$ )和微观  $F_1$  值(Micro- $F_1$ )。宏观  $F_1$  值的计算方式:首先需要根据公式(12-1)分别计算每个类别的  $F_1$ ,然后再根据公式(12-2)来计算它们的平均值,即为宏观  $F_1$  值。此外,宏观  $F_1$  值还有一种计算方式:首先计算  $p$  和  $r$  的平均值,然后代入公式(12-1)来求宏观  $F_1$  值,这个过程可以用公式(12-3)来表示,本文将采用这种方式来计算分类器的宏观  $F_1$  值。微观  $F_1$  值的计算方式:首先需要在整个测试网页集合内分别统计  $p$  和  $r$  的值,然后根据公式(12-1)计算微观  $F_1$  值。

$$F_1 = \frac{2pr}{p+r} \quad (12-1)$$

$$\text{Macro-}F_1 = \frac{1}{m} \sum_{i=1}^m F_{1i} \quad (12-2)$$

$$\text{Macro-}F_1 = \frac{2 \times \sum_{i=1}^m p_i \times \sum_{i=1}^m r_i}{\left( \sum_{i=1}^m p_i + \sum_{i=1}^m r_i \right) \times m} \quad (12-3)$$

其中, $p$  为查准率; $r$  为查全率; $m$  为训练集类别数,这里为 12。虽然在使用的分类体系中共包含 733 个类别(样本集中类别及实例数量的分布情况详见表 12-1),但是为简单起见,把子类的分类结果分别统计到 12 个大类中,所以最后共有 12 个类的分类统计结果。

对于  $F_1$  值,从公式(12-3)可以看出,它反映了查准率  $p$  和查全率  $r$  之间的平衡关系;只有当  $p$  和  $r$  比较接近,并且取值都比较大时, $F_1$  才比较大。反之,当  $p$  和  $r$

相差比较悬殊,或者取值都比较小时, $F_1$  值就比较小。所以, $F_1$  综合反映了分类器的整体性能。本章将使用宏观  $F_1$  值和微观  $F_1$  来评价分类器的质量。

## 二、训练样本

为了推进信息检索领域的发展,由美国国家标准和技术研究院(NIST)、信息技术实验室(ITL)检索小组、美国国防部高级研究计划署(DARPA)信息技术处、高级研究开发机构(ARDA)等单位共同发起了有全球影响的信息检索会议 TREC,自 1992 年起每年一次;TREC 会议实际上是文本信息检索系统的擂台赛,可以说,在 TREC 上展示的文本分类系统代表了文本分类领域的最新研究成果。一些大学,如 CMU、BERKLEY、CORNELL 等和一些公司带着自己开发的文本分类系统参加会议,由大会使用相同的训练集和测试集对这些系统进行评测。中国科学院计算所、清华大学、复旦大学等单位近几年也有派队参加,并取得了不错的成绩。同时我们注意到,由于 Web 技术的发展,TREC 也逐步开始提供标准的英文网页语料来评测 Web 信息检索系统。

与面向英文的分类系统相比,中文分类系统的起步比较晚。从第五次 TREC 会议开始,增加了对中文分类系统的评测。实际上参加 TREC-5 的中文分类系统处理的重点还停留在中文的分词问题上,而且处理的对象还是新华社的新闻稿这类普通的中文文本。基于案例的有指导的机器学习方法是实现中文网页自动分类的理论基础。因此,中文网页训练集是实现中文网页自动分类的前提条件。但是,到目前为止,还没有出现标准的中文网页语料库,因此也没有出现针对中文网页分类系统的评测。为了解决这一问题,我们通过动员不同专业的几十个学生,人工选取形成了一个基于层次模型的大规模中文网页样本集<sup>①</sup>。它包括 12 336 个训练网页实例和 3269 个测试网页实例,分布在 733 个类别中,每个类别平均有 17 个训练实例和 4.5 个测试实例。样本集中类别及实例数量的分布情况如表 12-1 所示。此外,为了搜集网页的方便,开发了一个网页实例集的搜集和整理的工具 WebSmart,如图 12-4 所示。经过实际应用的检验,表明该工具操作方便,使用它可以非常方便地实现网页实例集的搜集和整理。

表 12-1 样本集中类别及实例数量的分布情况表

类别编号	类别名称	类别数	训练样本数	测试样本数
1	人文与艺术	24	419	110
2	新闻与媒体	7	125	19
3	商业与经济	48	839	214
4	娱乐与休闲	88	1510	374

<sup>①</sup> 天网免费提供网页样本集给有兴趣的同行,燕穹产品号:YQ-WEBENCH-V0.8。

续表

类别编号	类别名称	类别数	训练样本数	测试样本数
5	计算机与因特网	58	925	238
6	教育	18	286	85
7	各国风情	53	891	235
8	自然科学	113	1892	514
9	政府与政治	18	288	84
10	社会科学	104	1765	479
11	医疗与健康	136	2295	616
12	社会与文化	66	1101	301
共 计		733	12336	3269

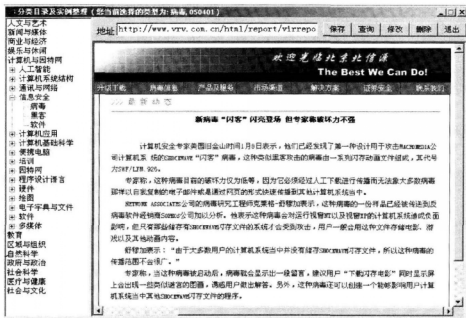


图 12-4 WebSmart——一个网页实例集搜集和整理工具

下面简要地介绍上述中文网页样本集的分类体系。长期以来,国内外已存在一些可以借鉴的信息内容分类标准,通过比较分析它们的特点和对中文网页这一新型分类对象的适应性,提出了我们的分类体系。

国外具有代表性的分类标准有:《杜威十进分类法》、《美国科研系统常用分类法》、《联合国教科文组织大学学科分类法》等。由于文化背景、思维习惯等方面的不同,这些标准不能完全适应中文文献的分类。例如,分类名称和涵义存在着差异,国内多数分类法把人文科学包含在社会科学(哲学社会科学)领域内,而国外通常将社会科学和人文科学加以区别。





数等于最右边的 19 时,表示训练集中的所有样本都被保留下来,而不是为每个类别取 19 个训练样本)。

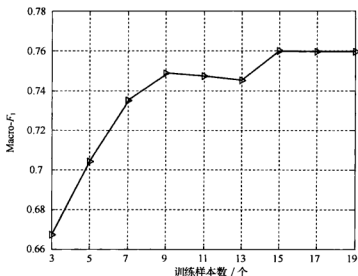


图 12-6 Macro-F<sub>1</sub> 值随样本数的变化

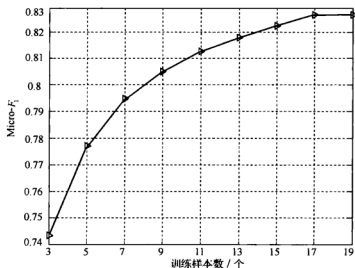


图 12-7 Micro-F<sub>1</sub> 值随样本数的变化

从图 12-6 可以看出,当训练样本数大于等于 15 时,分类器的宏观  $F_1$  值就稳定下来了,尽管此时图 12-7 的微观  $F_1$  还在上升,但是增加的幅度已经很小了,并且到 17 时平稳下来。因此,针对本文的训练集,最小训练样本数取 15 个。

### 三、特征选取

实现文本自动分类的基本困难之一是特征项空间的维数过高。所谓“特征项”在中文文本中主要指分词处理后得到的词汇,而特征项的维数则对应不同词汇的个数。数量过大的特征项一方面导致分类算法的代价过高,另一方面导致无法准确地提取文档的类别信息,造成分类效果不佳。因此,需要在不牺牲分类质量的前提下尽可能地降低特征项空间的维数。“特征选取”的任务就是要将信息量小,“不重要”的词汇从特征项空间中删除,从而减少特征项的个数,它是文本自动分类系统中的一个关键步骤。

为便于后面的描述,这里简要给出特征选取的一般过程。给定训练文档集合  $D = \{d_1, d_2, \dots, d_n\}$ , 设  $T = \{t_1, t_2, \dots, t_m\}$  为对  $D$  中的文档做分词后得到的词汇全集, 用  $[m]$  表示集合  $\{1, 2, \dots, m\}$ 。所谓“特征选取”可以看成是确定从 TERMS 到  $[m]$  的一个 1-1 映射, 即

$$F\text{-Selection: } T \rightarrow [m]$$

然后根据计算开销的考虑, 取一个  $i \in [m]$ , 认为  $T$  中那些函数值不小于  $i$  的词汇为“选取的特征项”, 记做  $T_i$ 。

在完成了特征选取后, 分类就是基于  $T_i$ , 即以其中的元素为基础, 用一个向量来表达每一个文档。分类的过程就是按照某种算法来比较待分类文档的表示向量和训练集文档的表示向量, 取最相近者所处的类为待分类文档的类。

文献(Yang et al. 1997)研究了多种特征选取方法, 如文档频率(Document Frequency, DF)、信息增益(Information Gain, IG)、互信息(Mutual Information, MI)、开方检验( $\chi^2$  test, CHI)、术语强度(Term Strength, TS)等。针对英文纯文本比较研究了上述五种经典特征选取方法的优劣。实验结果表明: CHI 和 IG 方法的效果最佳; DF 方法的性能同 IG 和 CHI 的性能大体相当, 而且 DF 方法还具有实现简单、算法复杂度低等优点; TS 方法性能一般; MI 方法的性能最差。针对中文网页, 其结论是否还正确, 目前还没有很明确的结论。因此, 本节使用同一个中文网页数据集评测了 DF、IG、MI 以及 CHI 等四种常见的特征选取方法。下面对这些典型的特征选取算法做一下简单地介绍。

#### (1) 文档频率

DF 表示在训练集中包含某个特征项  $t$  的文档数。这种衡量特征项重要程度的方法基于这样一个假设: DF 较小的特征项对分类结果的影响较小。这种方法优先取 DF 较大的特征项, 而 DF 较小的特征项将被剔除。即特征项按照 DF 值排序。这里, 为物理意义清楚起见, 并没有像本节开始那样严格的从 TERMS 到  $[m]$  的映射, 但显然这是没有困难的, 不赘述(后同)。不过我们注意到, 这种策略不符合被广泛接受的信息检索理论: 高频词没有低频词对文档特征贡献大(Yang et al. 1997)。DF 是最简单的特征项选取方法, 而且该方法的计算复杂度低, 能够胜任大规模的分类

任务。

## (2) 信息增益

IG 通过统计某个特征项  $t$  在一篇文档中出现或不出现的次数来预测文档的类别。IG 的计算公式如公式(12-4)所示(Yang et al. 1997)。

$$G(t) = - \sum_{i=1}^m P_r(c_i) \lg P_r(c_i) + p_r(t) \sum_{i=1}^m P_r(c_i | t) \lg P_r(c_i | t) \\ + P_r(\bar{t}) \sum_{i=1}^m P_r(c_i | \bar{t}) \lg P_r(c_i | \bar{t}) \quad (12-4)$$

其中,  $P_r(c_i)$  表示一篇文档属于类别  $c_i$  的概率;  $P_r(t)$  表示特征项  $t$  在一篇文档内出现的概率;  $P_r(\bar{t})$  表示特征项  $t$  不在一篇文档内出现的概率;  $P_r(c_i | t)$  表示特征项  $t$  在属于类别  $c_i$  的文档内出现的概率;  $P_r(c_i | \bar{t})$  表示特征项  $t$  不在属于类别  $c_i$  的文档内出现的概率。 $m$  是文档类别数。 $G(t)$  值大则被选取的可能性大, 即特征项按照  $G$  值排序。

## (3) 互信息

MI 使用公式(12-5) 计算某个特征项  $t$  和类别  $c$  之间的相关性(Yang et al. 1997)。

$$I(t, c) \approx \lg \frac{A \times N}{(A + C) \times (A + B)} \quad (12-5)$$

其中,  $A$  为  $t$  和  $c$  同时出现的次数;  $B$  为  $t$  出现而  $c$  没有出现的次数;  $C$  为  $c$  出现而  $t$  没有出现的次数。  $N$  为所有文档数。如果  $t$  和  $c$  不相关, 则  $I(t, c)$  值为 0。如果有  $m$  个类, 于是对于每个  $t$  会有  $m$  个值, 取它们的平均, 就可得到特征选取所需的一个线性序。大的  $I$  平均值的特征被选取的可能性大。

## (4) CHI

使用 MI 衡量特征项的重要程度时, 只考虑到了正相关对特征项重要程度的影响。如果特征项  $t$  和类别  $c$  反相关, 就说明含有特征项  $t$  的文档不属于  $c$  的概率要大一些, 这对于判断一篇文档是否不属于类别  $c$  也是很有指导意义的。为克服这个缺陷, CHI 使用公式(12-6) 计算特征项  $t$  和类别  $c$  的相关性。

$$\chi^2(t, c) = \frac{N \times (AD - CB)^2}{(A + C) \times (B + D) \times (A + B) \times (C + D)} \quad (12-6)$$

其中,  $A$  为  $t$  和  $c$  同时出现的次数;  $B$  为  $t$  出现而  $c$  没有出现的次数。  $C$  为  $c$  出现而  $t$  没有出现的次数;  $D$  为  $t$  和  $c$  同时没有出现的次数。  $N$  为训练集中的文档数。与 MI 类似, 如果  $t$  和  $c$  不相关, 则  $\chi^2(t, c)$  值为 0。同 MI 相同, 如果有  $m$  个类, 每个  $t$  就会有  $m$  个值, 取它们的平均, 就可得到特征选取所需的一个线性序。大的  $\chi^2$  平均值的特征被选取的可能性大。

为了观察特征项的个数对分类器性能的影响, 分别使用 CHI、IG、DF、MI 特征选取算法挑选出不同个数的特征项。图 12-8 和图 12-9 分别表示当取不同百分比的特征项时, 分类器宏观  $F_1$  和微观  $F_1$  的变化。这里需要说明的是, 最右边的 45% 表示的所有特征项都被保留下来, 而不是仅保留前 45% 个特征项。

从图 12-8 可以看出,CHI 方法最优,在取前 15% 的特征项时,分类器的质量就稳定下来了;使用 DF 方法时,分类器的宏观  $F_1$  波动最小,可以过滤掉 80% 的特征项;MI 方法和 IG 方法都在取前 25% 的特征项时,分类器的质量才稳定下来了。

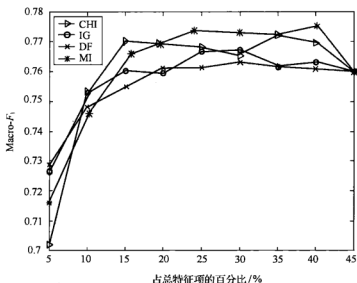


图 12-8 CHI、IG、DF、MI 的比较 (Macro- $F_1$ )

从图 12-9 可以看出,IG 方法最优,在取前 15% 的特征项时,分类器的质量就稳定下来了;使用 DF 方法时,分类器的宏观  $F_1$  波动最小,可以过滤掉 80% 的特征项;MI 方法最差,在取前 25% 的特征项时,分类器的质量才稳定下来了。

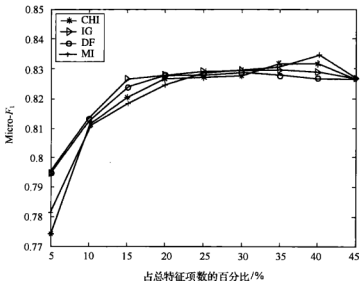


图 12-9 CHI、IG、DF、MI 的比较 (Micro- $F_1$ )

综合图 12-8 和图 12-9,可以看出,CHI、IG 和 DF 的性能明显优于 MI;CHI、IG 和 DF 的性能大体相当,都能够过滤掉 80% 以上的特征项;DF 具有算法简单、质量高的优点,可以用来代替 CHI 和 IG,但是同被广泛接受的信息检索理论有些矛盾。我们这里得到的结论,同文献(Yang et al. 1997)使用普通英文文本评测结果基本一致。

#### 四、分类算法

在本章第二节,有了一个关于各种文档自动分类算法的概貌。下面对几个比较典型的分类算法进行具体的介绍,并给出了 kNN 与 NB 算法的分类质量与效率的实验结果比较。

##### 1. 典型分类算法

##### (1) kNN 分类算法

kNN 分类算法是一种传统的基于统计的模式识别方法。算法思想很简单:对于一篇待分类文档,系统在训练集中找到  $k$  个最相近的邻居,使用这  $k$  个邻居的类别为该文档的候选类别。该文档与  $k$  个邻居之间的相似度按类别分别求和,减去一个预先得到的截尾阈值,就得到该文档的类别测度。用 kNN 也表示所选  $k$  个最相近文档的集合,公式(12-7)刻画了上述思想(Yang et al. 1999)。

$$y(\mathbf{x}, c_j) = \sum_{d_i \in \text{kNN}} \text{sim}(\mathbf{x}, d_i) y(d_i, c_j) - b_j \quad (12-7)$$

其中,  $\mathbf{x}$  为一篇待分类网页的向量表示;  $d_i$  为训练集中的一篇实例网页的向量表示;  $c_j$  为一类别;  $y(d_i, c_j) \in \{0, 1\}$  (当  $d_i$  属于  $c_j$  时取 1; 当  $d_i$  不属于  $c_j$  时取 0);  $b_j$  为预先计算得到的  $c_j$  的最优截尾阈值;  $\text{sim}(\mathbf{x}, d_i)$  为待分类网页与网页实例之间的相似度,由文档间的余弦相似度公式(12-8)计算得到

$$\cos(\mathbf{x}, d) = \frac{\mathbf{x} \cdot d}{\|\mathbf{x}\| \|d\|} \quad (12-8)$$

kNN 算法本身简单有效,它是一种 lazy-learning 算法,分类器不需要使用训练集进行训练,训练时间复杂度为 0。kNN 分类的计算复杂度和训练集中的文档数目成正比,即如果训练集中文档总数为  $n$ ,那么 kNN 的分类时间复杂度为  $O(n)$ 。

##### (2) NB(Naïve Bayes)算法

NB 算法是基于贝叶斯全概率公式的一种分类算法。贝叶斯全概率公式的定义如公式(12-9)所示(Sebastiani 1999)。

$$P(A | B) = \frac{P(B | A)P(A)}{P(B)} \quad (12-9)$$

给定一个类  $c$  以及文档  $d(a_1, a_2, \dots, a_n)$ , 其中  $a_i$  表示文档  $d$  中出现的第  $i$  个特征项的权值,  $n$  为文档中出现的特征项的总数。根据全概率公式,可以得到

$$P(c|d) = \frac{P(d|c)P(c)}{P(d)} = \frac{P(a_1|c)P(a_2|c)\cdots P(a_n|c)P(c)}{P(d)} \quad (12-10)$$

其中,  $P(c|d)$  表示文档  $d$  属于类别  $c$  的概率;  $P(c)$  表示待分类的文档所处的领域中文档属于这个类的概率, 在具体的计算时, 可以分别用训练集中属于这个类的文档所占的比例代替。  $P(a_i|c)$  表示在类别  $c$  中特征项  $a_i$  出现的概率, 可以近似地用训练集中包含有该特征项的类别  $c$  中的文档个数与训练集中类别为  $c$  的文档总数的比值表示。

由此可以看出, NB 算法假设文档之间的特征项都是相互独立的。但是, 这一假设对语义丰富的语言文字信息往往过于简单, 这也在一定程度上限制了算法的性能。

NB 算法需要使用训练集对分类器进行训练, 即需要分别计算每个  $P(a_i|c)$ 。假设训练集共有  $m$  个类别,  $n$  个特征项, 待分类文档共有  $k$  个特征项, 那么训练的时间复杂度为  $O(m * n)$ 。分类的时间复杂度为  $O(k)$ 。

### (3) 决策树(Decision Tree, Dtree)算法

决策树算法通过对训练数据的学习, 总结出一般化的规则, 然后再利用这些规则解决问题。用决策树进行文档分类的基本思路是这样的: 先用训练集为预先定义的每一个类构造一棵决策树, 构造方法如下:

1) 以训练集作为树的根节点, 它表示所有的训练文档, 将它标记为“未被检测”。

2) 找到一个标记为“未被检测”的叶节点, 如果它表示的所有文档都属于这个类, 或者都不属于这个类, 将这个叶节点的标记改为“已被检测”, 然后直接跳到第三步; 否则, 挑选当前最能区分这个节点表示的文档集中属于这个类的文档和不属于这个类的文档的特征项作为这个节点的属性值, 然后以这个节点为父节点, 增添两个新的叶节点, 都标记为“未被检测”, 父节点表示的训练文档集中含有这个特征项的所有文档用左子节点表示, 所有不含有这个特征项的文档用右子节点表示。

3) 重复第二步操作, 直到所有的叶节点都被检测过。

对每棵决策树, 从它的根节点开始, 判断节点的属性值(特征项)是否在待分类的文档中出现, 如果出现, 则沿着左子树向下走; 否则沿着右子树向下, 再继续判断当前节点的属性值是否在待分类的文档中出现, 直到到达决策树的某个叶节点, 如果这个叶节点表示的训练文档都属于这个类, 则判定这篇待分类的文档也属于这个类; 反之亦然。

### (4) Rocchio 算法

其基本思想是使用训练集为每个类构造一个原型向量, 构造方法如下: 给定一个类, 训练集中所有属于这个类的文档对应向量的分量用正数表示, 所有不属于这个类的文档对应向量的分量用负数表示, 然后把所有的向量加起来, 得到的和向量就是这个类的原型向量, 定义两个向量的相似度为这两个向量夹角的余弦, 逐一计算训练集中所有文档和原型向量的相似度, 然后按一定的算法从中挑选某个相似度作为界。给定一篇文档, 如果这篇文档与原型向量的相似度比界大, 则这篇文档属于这个类, 否则这篇文档就不属于这个类。Rocchio 算法的突出优点是容易实现, 计算(训练和

分类)特别简单,它通常用来实现衡量分类系统性能的基准系统,而实用的分类系统很少采用这种算法解决具体的分类问题。

## 2. kNN 与 NB 算法比较

文献(Yang et al. 1999)比较研究了支持向量机(SVM)、kNN、NB、Linear Least Squares Fits(LLSF)和 Neural Network(NNet)算法。研究结果表明,当训练集中每个类的正面实例比较少(少于10个)的情况下,SVM、kNN、LLSF的性能明显优于NNet和NB算法。这里,将比较研究kNN和NB算法。

从表12-2可以看出,kNN的质量明显优于NB算法,但是,NB算法的分类效率要比kNN方法略高。kNN和NB分类算法对12个不同类别的分类情况如图12-10所示。

表 12-2 kNN 和 NB 算法的分类质量和分类效率比较

指 标 算 法	质 量		效率/秒	
	Micro- $F_1$	Macro- $F_1$	训练时间	测试时间
kNN	0.8266	0.7560	0	2426
NB	0.1934	0.1612	251	2129

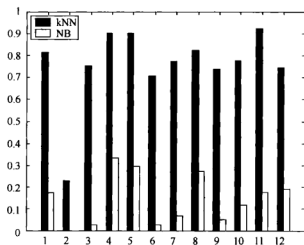


图 12-10 kNN 与 NB 分类结果的比较

从图12-10可以看出,kNN分类算法明显优于NB算法,对于所有类别,kNN的分类结果都优于NB。从图12-10还可以看出,即使是同一个算法对于不同类别的文档,其分类能力也是各有差异的。在“医疗与健康”领域,kNN的Macro- $F_1$ 达到最高值;在“新闻与媒体”领域,kNN的分类Macro- $F_1$ 达到最低值。对于“医疗与健康”领域,NB的Macro- $F_1$ 达到最高值;在“计算机与因特网”领域,NB的Macro- $F_1$ 达到最低值。从总体而言,NB算法对不同类别比较敏感,是一种不稳定的分类算法。kNN



的分类质量受领域的影响不大。

针对 kNN 分类算法,分析了三种因素对分类质量的影响; $k$  的取值;衡量两篇文档之间相似度的方法;兰式距离法(Canberra metric)与欧式距离法(Euclidean distance);分类目录中类别之间的层次关系。下面首先介绍  $k$  的取值对分类质量的影响。

### (1) $k$ 的取值

在前文的实验中,一直取  $k=20$ ,但这不一定是  $k$  的最好取值。因此,这里将通过具体实验来验证  $k$  的取值对分类质量的影响。

从图 12-11 和图 12-12 可以看出,当  $k=15$  时,分类器具有最佳的分类质量,随后,随着  $k$  值的增加,分类器分类质量有所下降,最后平稳下来。

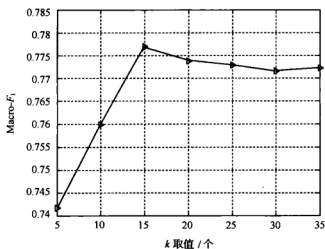


图 12-11  $k$  的取值对分类器质量的影响(Macro- $F_1$ )

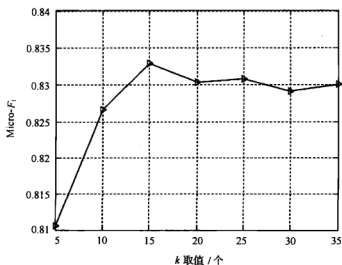


图 12-12  $k$  的取值对分类器质量的影响(Micro- $F_1$ )

## (2) 相似度的度量

为了计算两个文档之间的相似度可以有多种距离函数(卜东波 2000),其中欧式距离和兰式距离在实际的分类系统中用得较多,通常使用的两个向量之间夹角的余弦就是兰式距离的一种。

从表 12-3 可以看出,无论是分类质量,还是分类效率,应用兰式距离法来度量两个文档之间的相似度的分类效果明显优于欧式距离法。兰式距离法与欧式距离法对 12 个不同类别的分类情况如图 12-13 所示。

表 12-3 欧式距离与兰式距离的比较

算 法	质 量		效率/秒
	Micro- $F_1$	Macro- $F_1$	测试时间
欧式距离	0.2419	0.1715	4790
兰式距离	0.8266	0.7600	2426

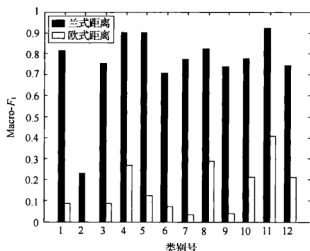


图 12-13 兰式距离法与欧式距离法对 12 个不同类别的分类情况

## (3) 分类目录中类别之间的层次关系

本节采用的分类体系包含 3 个层次,12 个大类,共 733 个类别(图 12-5),在分类过程中,可以利用这种层次关系来减少文档比较的次数:当某篇待分类的文档与父类文档之间的相似度小于某个阈值时,就可以认为与该父类包含的子类是不相关的,因此可以直接减少文档比较的次数。表 12-4 给出了应用于层次模型的 kNN 与基本 kNN 的分类质量和分类效率的比较情况。从表 12-4 可以看出,这种基于层次模型的 kNN 分类方法的效率是基本 kNN 分类方法的 3 倍多,但是,分类质量也下降了许多。图 12-14 给出了两者在 12 个大类下的分类情况比较。

表 12-4 基于层次模型的 kNN 与基本 kNN 的比较

算 法 \ 指 标	质 量		效率/秒
	Micro- $F_1$	Macro- $F_1$	测试时间
基于层次的 kNN	0.6723	0.5719	792
基本 kNN	0.8266	0.7600	2426

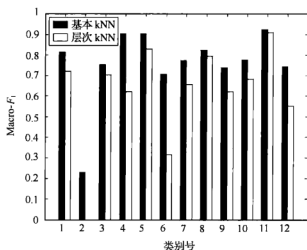


图 12-14 基于层次模型的 kNN 与基本 kNN 的比较

## 五、截尾算法

对于一篇待分类文档,应用  $m$  元分类算法通常得到多个类别。一般情况下都要求从这些候选类别中选择部分类别为该文档的最终分类结果。这个过程使用的方法通常被称为阈值策略。下面简单介绍三个比较常见的阈值策略(Yang 2001)。

### 1. 位置截尾法(rank-based thresholding, RCut)

假设分类系统预先定义类别数为  $m$ 。整数  $k$  大于 1 并且小于  $m$ 。对于每一个待分类的文档  $D$ ,分类系统都返回一个长为  $m$  的候选类列表,取候选类列表的前  $k$  项(按类和文档的相似度排序),这篇文档就被认为属于这  $k$  个类。这种阈值策略就被称为位置截尾法。RCut 方法的优点是实现非常简单,能够胜任在线分类工作。但它存在严重的缺陷:假设待分类的文档数目为  $n$ ,候选类列表的每个位置都对应  $m$  个候选类。即使  $k$  变化 1,每篇文档的类关系都要发生变化。因此,无法平滑地调整分类系统的性能。称 RCut 算法是以文档为中心的。

### 2. 比例截尾法(proportion-based thresholding, PCut)

假设待分类的文档数目为  $n$ ,预先定义类别数为  $m$ 。 $P_i$  表示训练集中属于类  $i$

的文档所占的比例。系统首先计算出每篇待分类文档的候选类列表,然后生成每个类的候选文档列表(按类和文档的相似度排序)。对于类  $i$ ,取这个类的候选文档列表中的前  $n * P_i * x$  篇文档属于这个类,其他的文档则不属于这个类。其中  $x$  是经验比例因子(为实数),通过改变它的大小,可以平滑地调整系统的性能。PCut 算法的基本思想是控制分入各个类的文档数,使它们保持训练集中各个类文档数的比例关系。这种算法最大的问题是过分依赖于比例关系,而没有考虑类在候选类列表中的位置。可以看到,PCut 算法是以类别为中心的。与 RCut 算法相比,PCut 算法的系统性能比较平滑,但是不适用于在线分类。

### 3. 最优截尾法(score-based local optimization thresholding, SCut)

同 PCut 算法一样,SCut 算法也是以类别为中心。假设待分类的文档数目为  $n$ ,预先定义的类别数为  $m$ 。系统首先计算出每篇待分类文档的候选类列表,然后生成每个类的候选文档列表(按类和文档的相似度排序)。对于候选类列表里的每一个类,如果这篇文档和这个类的相似度大于这个类的最优截尾相似度,那么这篇文档就属于这个类。否则,这篇文档就不属于这个类,其中,每个类的最优截尾相似度是这样预先取得的:将训练集分成两部分,其中一部分仍然作为训练集,另一部分作为测试集,对每一个类,评价分类系统在这个测试集下对于这个类的分类性能,调整截尾相似度,使得系统的性能达到最优,此时截尾相似度的值就是这个类的最优截尾相似度。SCut 算法性能比较优异,但是不能很好地处理那些稀有类别(就是比较少见的类别)。

文献(Yang 2001)比较研究了上述三种阈值策略,结果发现 SCut 算法效果明显优于 PCut 和 RCut 算法。由于本文使用的训练样本分布比较均匀,每个类平均有 17 个训练网页,对于这种基本按比例分布的样本集,PCut 方法就没有什么作用了,因此,比较研究了 RCut 和 SCut 方法,总体分类结果如表 12-5 所示。

表 12-5 RCut 和 SCut 截尾算法的比较

指 标 算 法	质 量		效率/秒
	Micro- $F_1$	Macro- $F_1$	测试时间
RCut	0.8266	0.7600	4324
SCut	0.8401	0.7849	5368
基本 kNN	0.8266	0.7600	2426

从表 12-5 可以看出,SCut 方法比 RCut 方法在分类质量上要,而分类效率却要差些,但是两者的差别不是十分明显。这里,RCut 方法的分类质量同基本 kNN 方法的分类质量完全一样,因为通过实验测试发现,当  $R=1$  时,分类器的分类质量最好,RCut 的这种取大的一个文档类别的计算方法同普通 kNN 的计算方法一样。所以,两者的分类结果是一样的。具体达到 12 个大类,两者分类结果的比较如图 12-15 所示,从中可以看出,SCut 比 RCut 方法的效果要好一些。

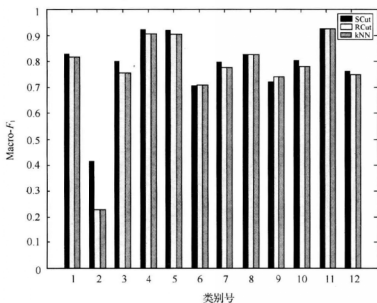


图 12-15 RCut 和 SCut 截尾算法的比较

## 六、中文网页分类器的设计方案

在上一小节定量地分析了影响分类器性能的各种关键因素。据此,我们设计了一个分类器,具体的要素和参数如表 12-6 所示。

表 12-6 一个分类器的设计方案

关键因素		方 案
训练样本数		15
特征选取方法		CHI
分类 算法	kNN & NB	kNN
	$k$	15
	相似度	兰式
	层次关系	层次关系
截尾算法		SCut

## 第四节 天网目录导航服务

### 一、问题的提出

目前提供 Web 浏览导航的系统主要分为两大类。第一类是目录导航系统。它主要是通过具有专业知识的网页编辑人员人工地对网站进行精选,建立一个内容分类索引目录,向用户提供目录导航服务。用户可以沿着分类目录的层次结构,逐步进入

自己感兴趣的主题,进而找到所需的信息。这类系统的优点是提供的相关信息比例很高,而且信息内容的权威性较高,网页信息和用户的相关性较大,但是覆盖的范围小,系统维护的网页数量有限。其典型代表是 Yahoo! 的目录系统。第二类是基于自动信息搜集和分析的搜索引擎系统(称为“自动式搜索引擎”)。它通过一个程序自动地在网上沿着超文档链递归地访问、搜集 Web 网页,分析页面的内容,生成索引和摘要,并向用户提供 Web 查询界面,根据用户的查询关键词在索引库中查找相关信息在网上的位置,最后将查询结果按照相似度排序后返回,帮助用户尽快地找到所需的信息。这类系统的优点是涵盖的网页数量巨大,但搜索的准确率相对较低,其典型代表是 Google。

下面简要地比较自动式和目录式搜索引擎。

1) 自动式搜索引擎自动地搜集、分析和处理网页,因而它索引的网页数多,信息量大,并且能够定期或增量地搜集网页,更新索引库的内容,向用户提供最新的 Web 网页信息。但是它只提供基于关键词的检索,用户只有确切地知道自己感兴趣的网页含有哪些关键词时,查询的效果才比较理想。否则,返回的结果很可能和用户的实际需求“风马牛不相及”。

2) 目录式搜索引擎支持基于分类目录的查询。目录式搜索引擎对搜集的网页采用人工分类。由于人工方式对网页内容的理解比较准确,因此查询的准确性优于自动式搜索引擎。当用户对某个领域感兴趣但并不熟悉这个领域的关键词时,这种查询方式能很好地为用户提供服务。由于人工分类效率低,网页更新困难,目录式搜索引擎在索引的网页规模上受到很大的限制。Google 等自动式搜索引擎索引的网页数量早已突破十亿级,而 Yahoo! 要少得多。

天网搜索引擎是一种典型的自动式搜索引擎。下面介绍如何应用中文网页自动分类技术在天网中提供目录导航服务。

## 二、天网目录导航服务的体系结构

天网目录导航服务的体系结构如图 12-16 所示。系统整体上可以分为两个部分:在线部分和离线部分。在线部分首先接受用户的查询条件,然后根据用户所处的网页目录自动地进行查询扩展,接着在预先分好类的网页库中检索,最后实时地返回查询结果。离线部分的核心部件为中文网页分类器,负责中文网页的自动分类。本章使用的中文网页分类器的工作原理如图 12-3 所示。分类器的整个工作周期可以分成训练过程和分类过程。在训练过程中,训练集实例经过中文分词和特征选取处理后被表示成向量形式。该特征向量用来描述类别模式,在分类过程中使用。校验集是训练集的一部分,通过应用相应的阈值策略来预先确定每个类别的截尾阈值。在分类过程中,一个待分类的中文网页经过中文分词并表示成向量后,应用分类算法同训练过程得到的类别模式逐一比较,得到候选类别列表,然后同训练过程中得到的每个类别的阈值相比较,保留大于阈值的类别,并作为该网页的分类结果。

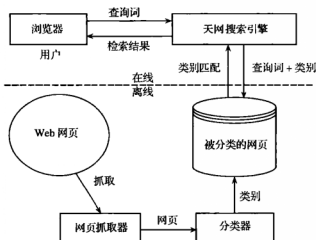


图 12-16 天网目录的体系结构

### 三、天网目录的运行实例

图 12-17 展示了天网目录导航服务的用户界面。通过目录导航服务,用户可以非常容易地查找他感兴趣的课题:通过三层目录,用户可以逐步缩小要检索内容的范围。实验测试表明,应用第四节介绍的分类型设计方案可以满足天网网页目录服务的要求。

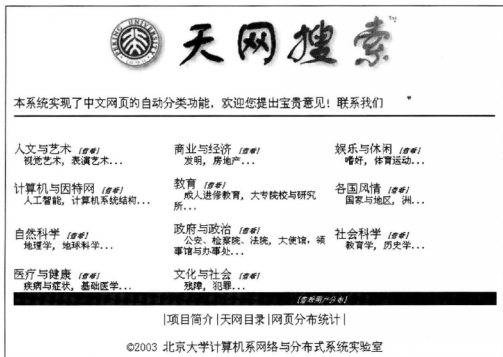


图 12-17 天网目录导航服务

## 第五节 文本聚类方法

在文本分类中,需要有一个事先给定的分类体系和训练样本集,如天网的中文网页分类体系,它将所有的网页分为 12 个大类,即任何网页的内容将属于这 12 个大类中某一个(或某几个)类别,训练样本集中的每一个网页被人工标记为属于某一类样本。这种利用已标记样本集对未知样本进行类别划分的方法称为“有监督的学习”(Supervised Learning)方法。本节所要介绍的聚类方法是一个“无监督的学习”(Unsupervised Learning)方法,它不需要提供训练数据,仅根据该组文本的内容特征而进行一种自然的划分。

### 一、文本聚类的一般过程

所谓文本聚类(Text Clustering),或称文档聚类(Document Clustering),就是将文本集合分成多个类或簇,使得在同一个簇中的文本内容具有较高的相似度,而不同簇中的文本内容差别较大。它可以看做是聚类分析技术与方法参照文献(刘远超等 2006)在文本信息处理领域中一种具体应用。

在信息检索中,文本聚类的早期应用是为了提高系统的查准率与查全率,并被用于寻找给定文本的相近文本。目前主要用于浏览文本、显示文本集合、组织搜索引擎的返回结果,如搜索引擎 Vivisimo 将查询结果进行聚类,这样有利于用户快速定位自己需要的信息。

在文本或网页聚类中,目前使用最多聚类算法是  $k$ -均值聚类与层次聚类方法,或是这两种基本方法的改进形式。其他的聚类方法包括自组织特征映射(Self-Organizing Map, SOM)、遗传算法等。

文本聚类的大致步骤概括如下:①对给定的文本集合;②对各个文本进行预处理,包括词法分析、英文的词干提取、中文分词或命名实体识别等;③对预处理后的各个文本分别进行特征选取或特征抽取等,构造文本集合的特征向量空间;④计算各个文本之间的相似度或距离,进而构造文本间的相异度(或相似度)矩阵;⑤选择某一种聚类算法进行聚类;⑥根据实际应用,进行类别选择与类别命名;⑦将聚类结果以适当的形式输出,如直接列表输出或以可视化的形式展示等;⑧对聚类结果进行评估,以检验聚类结果的质量。如结果不好,可以返回到前面的某一步进行重新聚类。对照前述的文本分类,也可将上述步骤称为文本聚类的一般过程,如图 12-18 所示。

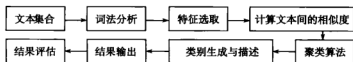


图 12-18 文本聚类的一般过程



若要对 Web 网页进行聚类,则应在数据的预处理步骤中,增加网页信息的去噪等处理环节,将其首先转化为纯文本信息。当然,也可考虑利用超链接、锚文本等网页中特有的信息,提高聚类的效果。

## 二、文本间相似性的度量

为了度量文本间的相似程度,需要定义一些用于划分类别的计量指标,常用的统计指标有距离和相似系数。“距离”属于相异性测度指标,它考察两个文本有多远。“相似系数”则属于相似性测度指标,它考察两个文本有多近。这两个指标从正反两个角度度量两个文本之间内容的差异性。显然,距离和相似系数成某种反比关系;常见的一种函数关系为  $\text{sim}(d_i, d_j) = 1/(1 + d_{ij})$ , 其中,  $\text{sim}(d_i, d_j)$  表示文本  $d_i$  与  $d_j$  间的相似系数,  $d_{ij}$  表示文本  $d_i$  与  $d_j$  间的距离。

对于有  $n$  个索引词(特征属性)的文本集合来说,  $m$  个文本就可以看做  $n$  维空间中的  $m$  个点,此时,每一个文本可以表示为一个  $n$  维向量。进而可以用点间的距离来度量文本间的距离。最常用的距离度量方法是欧几里得距离,定义为

$$d_{ij} = \sqrt{(w_{i1} - w_{j1})^2 + (w_{i2} - w_{j2})^2 + \cdots + (w_{in} - w_{jn})^2}$$

其中,第  $i$  篇文本与第  $j$  篇文本分别用  $(w_{i1}, w_{i2}, \dots, w_{in})$  和  $(w_{j1}, w_{j2}, \dots, w_{jn})$  表示;  $d_{ij}$  表示第  $i$  篇文本与第  $j$  篇文本间的距离。

文本间的相似系数有多种定义方法,包括余弦系数(Cosine Coefficient)、重叠系数(Overlap Coefficient)和雅可比系数(Jaccard Coefficient)等。使用较多的仍是余弦系数,即向量夹角的余弦值。

对于二值的情况相似系数的计算还有其他的度量指标。需要指出的是:文本之间不同度量指标的选择,所得到的聚类结果存在一定的差异,需要结合具体情况来考察哪些度量指标对哪些数据更有效。

## 三、常用聚类算法

目前,对数据进行聚类的方法已有很多种,主要包括层次聚类,划分方法,基于密度、基于网格、基于模型的方法等。多数方法都可用于文本聚类。以下我们介绍两个对文本聚类比较有效且使用较多的方法。

### 1. 层次聚类方法

层次聚类方法能对文本进行层次化的组织,即一个大的主题可以包含若干个小主题,形成一个目录层次结构,但该目录结构的表达不一定是精确的。

层次聚类方法通常用树图来表示其聚类过程,图 12-19 给出了一个文本聚类的例子。在树的最底层包含 5 个文本( $d_1, d_2, d_3, d_4, d_5$ )。在上一层中,聚类点 6 包含两个文本  $d_1$  和  $d_2$ , 聚类点 7 包含另两个文本  $d_4$  和  $d_5$ 。随着自下而上遍历该树,聚类的数目越来越少,用户可以选择查看在树的任意层次上的聚类。

层次聚类方法主要包括两类:合并的层次聚类方法与分裂的层次聚类方法(Han et al. 2007)。合并的(自底向上)层次聚类方法为:首先将每个文本看做一个类,然后相继合并相近的文本类,直到所有的文本合并为一个类,或者达到某个终止条件,如希望得到的类的个数或者两个相近的类超过了某一个阈值。分裂的(自顶向下)层次聚类方法为:首先将所有的文本看做一个类,然后逐渐细分为越来越小的类,直到每个文本自成一类,或者达到某个终止条件,如希望得到的类的个数或者两个相近的类超过了某一个阈值。

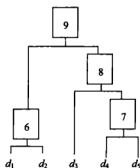


图 12-19 层次聚类实例

目前合并的层次聚类方法比分裂的层次聚类方法的应用更为广泛,算法的形式描述如下。

步骤 1:将文本集  $D$  中的每一个文本看做一个类。

步骤 2:计算  $D$  中每两个文本之间的距离。

步骤 3:Repeat。

步骤 4:将距离最小(或相似度最大)的两个文本类  $D_i, D_j$  合并为新的一类  $C$ ,其中  $D_i, D_j \in D$ 。

步骤 5:计算  $C$  与其他各类之间的距离。

步骤 6:until 所有文本合并为一类或达到某个终止条件。

在上述合并层次聚类算法的第 5 步,需要计算两类文本之间的距离或相似度,这是该算法执行过程的一个非常重要的环节,那么如何度量两类文本之间的距离或相似度呢?在实际应用中有多重定义方法,主要包括最短距离法、最长距离法、重心距离法、类平均距离法等(Han et al. 2007)。

层次聚类算法实质上是一种贪心算法。该方法的时间复杂度  $O(n^2)$ ,  $n$  为文本数。尽管层次聚类法比较简单实用,但存在一些问题;在聚类过程中,一旦一个步骤(合并或分裂)完成就不能进行修正,如果存在错误,此错误聚类将一直保持到聚类结束。目前已有多种改进方法,如凝聚层次的聚类与迭代重定位的集成等。

## 2. $k$ -均值聚类方法

$k$ -均值方法是一个简单高效、应用广泛的聚类方法,属于著名的划分聚类方法。给定一个包含  $n$  个文本的文本集合  $D$ ,以及要生成的分组个数  $k(k \leq n)$ ,一个划分聚类方法将文本集合  $D$  划分为  $k$  个分组,每一个分组代表一个聚类。典型的划分方法(Partitioning Methods)包括  $k$ -均值方法、 $k$ -中心点方法以及它们的变形。

利用  $k$ -均值方法进行聚类的大致过程是:首先随机选择  $k$  个对象作为初始值,用以代表一个簇的平均值或中心(类似于 Rochio 方法中的代表元),其中  $k$  是用户指定的参数,即所期望的簇的个数。对剩余的数据对象,根据它与各个簇中心的距离将它

赋给最近的簇。然后重新计算每个簇的平均值。这个过程不断重复,直至每个数据对象所属的簇不再发生变化(Han et al. 2007)。

基本的  $k$ -均值算法形式描述如下。

步骤 1: 任意选择  $k$  个文本作为初始的类的中心。

步骤 2: Repeat。

步骤 3: 根据类中文本的平均值, 将每个文本(重新)赋给最相近的类。

步骤 4: 更新类的平均值。

步骤 5: until 不再发生变化。

用一个实例说明上述算法的执行过程。假设一个文本集合  $D$  可用二维空间中的点表示, 如图 12-20(a) 所示。目标是将文本集  $D$  划分为两个聚类簇  $D_1, D_2$ , 即  $k=2$ 。首先任意选择两个文本作为初始的类的中心, 在图 12-20(b) 中用“+”表示。算法进入第一次 Repeat 循环。

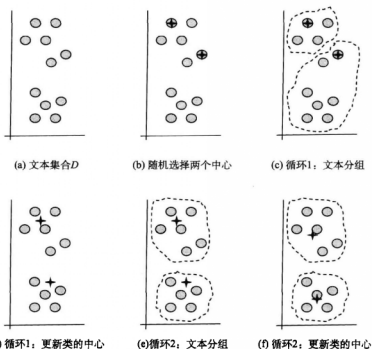


图 12-20  $k$ -均值算法进行文本聚类过程

循环 1: 每个文本被分配给离它距离最近的聚类中心, 形成两个聚类, 结果如图 12-20(c) 所示, 随后重新计算每一类的聚类中心, 结果如图 12-20(d) 所示。

循环 2: 在新的中心下, 再次将每个文本分配给离它距离最近的聚类中心, 又形成两个聚类, 结果如图 12-20(e) 所示, 接着又更新每一类的聚类中心, 结果如图 12-20(f) 所示。

再进行循环时,将发现没有文本被重新分配给不同的类,从而算法终止。

由于该算法在每次迭代时,都要计算每个小类的均值作为该类的代表元,因此该算法对异常数据十分敏感。该算法易于实现且效率较高,时间复杂度为  $O(tkn)$ ,其中,  $n$  为数据点的个数,  $k$  是聚类的个数,  $t$  是循环迭代的次数,由于  $k$  和  $t$  通常都远远小于  $n$ ,因此该算法相对于数据点数目而言常常是线性时间内可以完成的。

#### 四、聚类结果的评估

在使用某种聚类算法对文本集进行聚类后,如何对聚类结果进行评估是一件比较困难的事情,这与分类有着截然不同的不同,因为在对数据或文本聚类时事先并不知道它的正确聚类结果应该是什么。目前对聚类结果进行评估主要使用真实的分类数据集对聚类算法进行评估。此时,每一个数据对象都有一个类别标记,而每一个类别对应一个聚类,这样可以将聚类结果与实际的分类进行对比,以鉴别算法的好坏。例如,有三个类别的分类数据,利用分类算法得到三个聚类,可以通过比较类别的组成与实际类别的组成,以判断算法的优劣。此时,有多个指标可以度量聚类的质量,如信息熵、纯度、查准率、查全率和 F-score 等。如下给出信息熵和纯度这两个指标的具体含义,其他指标与前述检索结果及分类的指标是一样的。

设文本集  $D$  中共含有  $k$  类文本,记文本的类别集合为  $C = \{C_1, C_2, \dots, C_k\}$ ,某一聚类算法将文本集  $D$  聚为  $k$  个聚类(簇),即  $D$  被划分为两两不相交的集合  $D_1, D_2, \dots, D_k$ 。

信息熵(entropy):衡量聚类结果中每个簇由单个类文本的构成情况。具体地,对于聚类得到的每一个簇  $D_i$ ,首先计算该簇中各文本的类分布情况,即对于第  $i$  个簇,计算该簇的成员属于类  $j$  的概率(可用频率替代)  $p_{ij} = m_{ij}/m_i$ ,其中,  $m_i$  是第  $i$  个簇中文本的数量,  $m_{ij}$  是第  $i$  个簇中属于类  $j$  的文本数量。则第  $i$  个聚类簇的信息熵为,  $\text{entropy}(D_i) = -\sum_{j=1}^k p_{ij} \log_2(p_{ij})$ 。而整个聚类结果总的信息熵用每个簇的加权

和计算,即  $\text{entropy}(D_i) = \sum_{j=1}^k \frac{m_i}{m} \text{entropy}(D_i)$ ,其中  $m$  为文本集  $D$  中所含的文本总数。

纯度(purity):衡量聚类结果中每个簇由单个类文本构成情况的另一种度量指标。第  $i$  个簇的纯度定义为:  $\text{purity}(D_i) = \max_j (p_{ij})$ ;整个聚类结果总的纯度为每个簇纯度的加权和,即  $\text{purity}(D_i) = \sum_{j=1}^k \frac{m_i}{m} \text{purity}(D_i)$

需要指出的是,即使一个算法在某些已标注的数据集上有很好的聚类结果,这也不能保证它在实际的数据上也有好的聚类结果,这样的评估只是增加我们对该算法的聚类结果的期待而已。

## 五、搜索引擎返回结果的聚类

随着搜索引擎系统索引网页数量的快速增长,对用户给定的短查询,系统返回结果的数量也随之增多,如输入“搜索引擎”,Google 和百度都返回上亿条相关记录。这些结果以“线性列表”的形式呈现出来,用户需逐页查找与自己的需求相关的信息。此外,在自然语言中广泛存在着“一词多义”和“一义多词”的问题,用户也不一定能用一两个查询词清晰表达自己的信息需求。解决方法之一:对搜索引擎返回结果进行聚类,并给每个类别赋予一个反映该类信息特征的名字,这样搜索结果就被有效地组织起来,进而用户可以快速了解返回结果的整体分布情况,剔除无关信息,发现自己所需信息的类别,并在某一类信息中查找到自己所感兴趣的信息。这不仅更加精准,而且缩短了查询的时间(周登鹏 2007,奚婷 2008)。

搜索引擎返回结果聚类(Clustering Search Results)的方法,除使用常规的文本聚类算法外,使用较多的聚类算法有后缀树算法、Lingo 算法等,这两种算法简述如下。

1) 后缀树算法(Suffix Tree Clustering, STC)(E Zamir 1999, Weiss et al. 2001, 陈菊红 2009):不同于其他算法将文本看成是词的集合,该算法把文本看做有顺序的词串,并使用文本出现的短语串作为文本相似度的测度来构建后缀树,充分利用词与词之间的信息,后缀树中出现的相同的字符被认为是基本类,然后对基本类进行合并就可以得到最终的类。

后缀树算法主要包括两个步骤:提取基类和合并基类。首先,对输入的文本集合构造后缀树,其中后缀树的每个节点代表包含同一词组的文本集合,在后缀树中保留权重值大于某一阈值的节点作为基类,然后根据类相似度来构造基类关系图,最后将基类关系图的子图所对应的基类合并为一个最终类,由此便实现了文本聚类。

后缀树聚类算法的明显优势是使用词组为类别,提供简洁和有意义的类别描述,算法的时间复杂度与聚类的文本数线性相关。但是该算法中涉及的一些关键阈值很难再被调整;而且后缀树算法的词组修剪策略倾向于删除较长且包含较多信息量的词组,这种方式提取的类标签过于简单,无法准确的表达类别信息。

2) Lingo 算法(Osirisiki et al. 2004, 陈菊红 2009):目前多数聚类算法是从返回结果聚类的有效性方面进行考察,而基于奇异值分解的 Lingo 算法则主要考虑了类标签对聚类的重要作用。它首先根据抽象概念与关键词组的相似度提取类标签,然后再对文本进行分配。通过这样的处理可以提取出便于理解且充分反映类别内容的类标签。但是该算法的聚类结果没有层次性,不能很好地反映出文本数据之间的层次关系,不利于用户的查找。

对搜索引擎返回结果进行聚类的系统,以 Vivisimo(Searchenginewatch 2004)为最佳,它采用启发式算法来聚合文本信息,这种算法汲取了传统人工智能思想,强调对检索结果进行更好的描述和聚类,见第一章的介绍。另一个优点的是 Carrot2

系统,简述如下。

Carrot2 是基于 Java 开发的开源聚类搜索系统,主要用于对搜索结果进行聚类。与 Vivisimo 相似,它也是由用户输入关键字,然后从 Google、Bing 等大型搜索引擎系统中获取反馈结果,对其进行聚类,并通过树形的分类图进行显示,如图 12-21 所示。Carrot2 的聚类对象主要是各大搜索引擎返回的结果,通过文本聚类平台 Workbench 对结果进行聚类,并以 WebApp 方式将聚类结果呈现给终端用户。

Carrot2 采用的聚类算法主要是 Lingo 算法和 STC 后缀树聚类算法,目前 Carrot2 支持的聚类算法较多,代码开源可以进行版本的更新和改进。该系统应用广泛、可移植性较好,但是 Carrot2 中文分词效果和可视化效果欠佳。

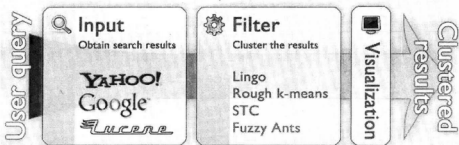


图 12-21 搜索结果聚类系统 Carrot2

随着 Web2.0 的蓬勃发展,如何帮助用户更好、更快地找到所需要的信息成为亟待解决的问题。相对于传统的搜索引擎,基于聚类方法的搜索引擎在解决该问题上具有更大的优势。聚类结果的智能化、结果呈现的可视化、用户参与的社会化及服务提供的个性化将是未来聚类搜索发展的方向。

## 第六节 小 结

本章重点讨论了中文网页自动分类的相关技术,比较了在同一个分类系统中不同环节上多种算法的优劣,得到了如下主要结论:

- 1) 对于规模比较大(如几百个类)的层次型网页内容分类体系,训练集中每一类的样本数达到 15 个时,得到分类器的分类质量就基本稳定了,再增加样本数对分类器质量的改进不大。
- 2) 在降低特征空间维度的各类特征选取算法中,IG、CHI 算法最有效;DF 等方法在一定程度上可以用来替代 IG 和 CHI;MI 算法最差。
- 3) 比较  $k$  最近邻居(kNN)和朴素贝叶斯(NB)分类算法,发现 kNN 的分类质量

明显优于 NB;而且 NB 算法对不同类别表现有明显差异,是一种不稳定的分类算法。但是,kNN 方法的分类效率要比 NB 差。

4) 三种因素对 kNN 算法的质量有影响。① $k$  的取值:当  $k$  取 15 时,分类器的分类质量最好。②衡量两篇文档之间相似度的方法:无论是分类质量,还是分类效率,兰式距离法都明显优于欧式距离法。③分类目录中类别之间的层次关系:基于层次的 kNN 分类算法的分类效率明显优于基本的 kNN 分类算法。但是,这需要牺牲一定的分类质量作为代价。

5) 比较 RCut 和 SCut 两种经典的截尾算法,实验表明 SCut 算法对分类器分类质量的影响明显优于 RCut 算法,但是算法费用要比 RCut 大。

这些研究成果系统地体现在天网提供的中文网页自动目录导航服务上。同时,作为本章各种实验研究基础的中文网页分类体系以及相应的训练测试集也是一项很好的工作成果。2003 年 3 月召开的首届“全国搜索引擎和网上信息挖掘学术研讨会”期间举行了由 10 个代表队参加的“中文网页自动分类竞赛”,该训练测试集为竞赛提供了有效的训练数据。

文本与网页聚类是 Web 信息组织的一种重要方式,其典型应用有效地解决了搜索引擎返回结果较多的问题。由于并未进行系统的实验研究,所以只在本章第五节做了简单的概述。

## 第十三章 开放域问答系统

本章讨论一种新的应用——开放域问答系统。和搜索引擎的人机交互方式不同,问答系统的查询不是关键词的组合,而是自然语言描述的问句,它的查询结果也不是一个文档集合,而是能够回答问句的自然语言描述。这样的人机交互方式比较符合用户的自然交流习惯,但也给系统设计和实现增加了难度。

本章将介绍开放域问答系统的体系结构、涉及的主要技术与方法。其中第一节介绍这项工作的发展历史、几个著名的系统和问答系统的基本体系结构。第二节至第四节比较详细地介绍问题分析模块、检索模块和答案抽取模块的主流方法和当前研究进展。第五节讨论基于常见提问问题库(FAQ)的问答式系统,特殊类问题回答的研究等。第六节讨论如何定量评测问答系统的性能。第七节介绍我们自行设计的一个英文问答系统,通过这个实例,使读者了解问答系统的设计和实现流程。

### 第一节 概 述

随着网络数据的快速增长,如何快速地从海量网络数据中获得相关信息是一个巨大的挑战,搜索引擎在一定程度上解决了这个问题。在分析搜索引擎日志时发现,它包含了许多自然语言表述的查询,如“如何安装 RedHat9”。这表明互联网用户更加习惯于用自然语言来表达他们的某些信息需求。搜索引擎的输入是一组关键词,但是有时用户的信息需求很难用关键词来确切表达。同时,用户所需信息的粒度有时并不是一篇文档,而是一个描述性的段落、句子、结论、人名或数字等,但是搜索引擎对于一个查询返回的是一个文档集合,用户还需从中找出相关的内容。这表明现有的搜索引擎服务和用户的实际信息需求之间存在两个方面的“鸿沟”:系统要求的关键词表达方式与用户自然表达方式之间的鸿沟和系统返回信息的方式与用户需要返回的方式之间的鸿沟。如果能使用户以一种更加自然的方式和系统交互,用户可以自然而精确地表达他们的信息需求,系统能直接返回用户想要知道的内容,就能填平这条鸿沟。基于这样的需求,开放域问答系统成为信息系统领域中继搜索引擎之后的又一个热点研究领域。从技术层面来看,计算机处理能力的提升,信息检索、自然语言处理、人工智能等相关领域研究的发展也为问答系统的构建创造了条件。

#### 一、问答系统的历史

早在计算机诞生不久的1950年,Alan Turing就提出了著名的图灵测试。该测



试的目的并不是为了获取信息,而是用于测试计算机是否具有智能,但是过程是相似的。图灵测试是把计算机和人都藏在用户看不见的地方,用户提出一系列的询问,计算机或者人给出问题的解答,如果用户分不清是人在回答还是计算机在回答问题,那么该计算机就具有了智能。为了鼓励进行图灵测试的研究,1991年 Hugh Loebner 设立了一个 Loebner Prize,奖金 10 万美元,用于奖励第一个通过图灵测试的系统,遗憾的是,迄今为止,尚没有个人或者组织能够获此殊荣。除了大奖以外,比赛还每年拿出 2000 美元奖励当年表现最出色的系统。十多年来,出现了 PC Therapist, Albert 等优秀的聊天机器人系统,它们提出的一些技术,很值得开放域问答系统所借鉴。

早期还有一些基于知识库的问答系统研究(Hendrix et al. 1978, Woods 1973),包括基于本体的问答系统,受限语言的数据库查询系统,问答式专家系统等。这些系统虽然能在特定的领域中达到比较好的性能,但是它们大多是受限的。首先是语言受限,即只能使用少数几种问题语言模式,一旦采用比较随意的语言,质量就会明显下降。其次是知识受限,一般只能够回答某一个特定领域中的专业性问题的。本章讨论的开放域问答系统和它们不同,具有两个特性:①它能够回答的问题不局限于一个或几个特殊的领域,而是不限定领域的;②它是基于一套文档数据库(可以是新闻集合,也可以是整个 Web),而且它只能回答那些答案存在于这个文档数据库中的问题。因此它是可扩展的,随着文档数据库的增加,它具有了更多的“知识”,就能回答更多的问题。

为了推动开放域问答系统的发展,信息检索评测组织(Text REtrieval Conference, TREC)自 1999 年开始,设立了开放域问答的评测任务,是 TREC 中历时最长的评测任务。其他的一些著名评测组织(如 NTCIR 和 CLEF)也设置了问答系统评测的任务。目前,关于问答系统的研究已在领域内受到强烈的关注。下面简要地介绍几个著名的问答系统。

## 二、著名开放域问答系统介绍

最早提供服务的问答系统是由美国麻省理工大学 Boris Katz 等开发 START 系统(<http://start.csail.mit.edu>,如图 13-1 所示),它于 1993 年 12 月对外开放,能够回答数以百万计的问题,包括位置、电影、人物、文化、历史、艺术、环境、词典定义等。它采用的主要技术被称为“自然语言注解技术”,即用句子或者短语作为“注解”来表达内容。另外,它还搜集了一些相关的知识性很强的网站内容构建他们的知识库。

AskJeeves 是另一个比较优秀的开放域问答系统(<http://www.ask.com>),和 START 不同,它返回的结果并不是一个精确的答案,而是可能包含答案的一系列段落,虽然用户要从段落中抽取答案需要一定的时间,但是有了上下文,用户对于答案的正确性会有更强的信心。该系统的另一个特色是它可以根据用户提问找一些临近的问题,当用户所提的问题表述不够准确或者存在歧义而无法获得正确答案时,系统

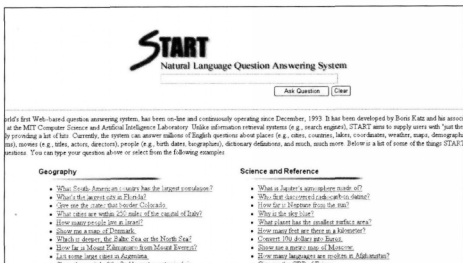


图 13-1 START 系统界面

的这种功能起到了很大的作用。下面以 who is the current president of China 为例在 AskJeeves 中查询, 获得的结果, 如图 13-2 所示。



图 13-2 Ask Jeeves 查询结果

比较著名的在线系统还包括 Brainboost(<http://www.answers.com>) 和 Answerbus(<http://www.answerbus.com>), 它们返回的是包含结果的句子。

### 三、开放域问答系统的通用体系结构

受限于自然语言处理、信息检索、人工智能等领域的现有技术和方法,问答系统

的回答能力是很有限的。文献(Moldovan et al. 2003)根据系统能力,把它由弱到强分成了5类。

1) 能回答事实问题的系统:回答的内容是一个事实,可以直接在文档里找到,一般是一个词或者一个词组。

2) 能回答具有简单推理问题的系统:回答的问题可能是文档里面的一个片断,需要系统简单的推理能力。

3) 能够多文档信息综合回答的系统:需要从多个文档中分别找出答案并且以一定的方式进行组合展示给用户。

4) 交互式问答系统:答案是上下文相关的,即和用户已经提问的问题和系统已经返回的结果有关系。

5) 具有类推能力的系统:答案需要系统推理获得,可能无法在文档集合中直接找到。

要实现第5类系统,需要很强大的知识库和推理能力,以现有的技术是很难实现的,所以暂不考虑。本章讨论的问答系统体系结构和模块主要针对第1~4类系统。

由于问答系统的发展尚不成熟,不同系统的体系结构存在着巨大的差异。如文献(Cui et al. 2004b)介绍的系统包括主题提取、主题定义、段落检索、答案抽取四个模块,文献(Wu et al. 2004)介绍的系统包括预处理、问题分析、答案类型分析、查询生成、答案生成这几个模块,文献(Xu et al. 2003)介绍的系统包括问句分类、文档检索、句子抽取、答案抽取、排序、查重等模块。

虽然不同系统的模块划分不同,但一般都包含三个大的部分,即问题的分析,文档和段落的检索和答案的提取和验证。曾经有一种观点认为一个问答系统就是由一个信息检索系统(IR,即对应于上面文档和段落的检索)和一个信息提取系统(IE,即对应于上面的答案提取),但这种提法过于简化问答系统的框架,在后续几节关键技术分析中,我们会发现,所需要的技术远远不止这两个领域。

问题分析部分所需要完成的功能包括问题中指代的消解替换,问句类型的分析,问题主题的识别,以及问题的语法分析等。在有上下文的问题中,很可能不出现一些实体名词,而用一些代词指代前面问题或答案里面出现过的实体,这时候需要取消这种指代,发现指向的原实体。问句分类是问答系统中一个很重要的环节,它需要把问句归到某一个类别中,以使得答案能够和这种问句类别相对应。在问答系统的解决方案中,有一部分研究者把回答准确率的提高寄希望于精细问句类型和精细实体的识别对应,所以他们尤其重视问句分类的性能。问题主题的识别,找出问题的主题可以帮助检索部分,先找出和主题相关的文档和段落,便于后续处理。在某些系统中,需要通过对问题进行语法分析,帮助检索模块和答案提取模块的工作。

文档和段落检索部分所需要完成的功能在于对一个问句,构造特定的查询,以便找到可能包含答案的文档或者段落。这里所涉及的问题包括:如何构造查询,采用什么样的信息检索模型,如何选择段落并且对这些段落进行排序,如何追求查全率和查准率之间的折衷,检索阶段的性能和最终的系统总性能有什么样的关系等。本章第

四节将详细讨论这些问题。

答案的提取和验证是最后一个部分,它分析检索获得的文档或者段落,从中提出能够回答问题的答案。在提取答案的时候第一部分所获得问题类型将有很大的影响,直接决定了提取怎样的答案。另外,某些问题的答案可能存在于知识库中或者 Web 上,这一步可以通过查看相应的知识库或者 Web 对答案进行验证。

三大模块之间的流程和关系如图 13-3 所示。

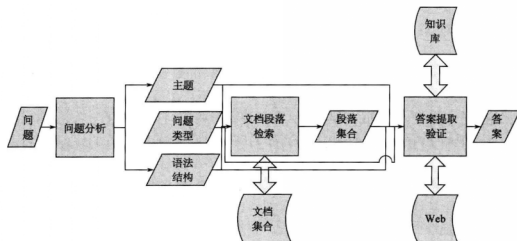


图 13-3 问答系统的通用体系结构

图 13-3 显示,问题分析模块可以获得问题的主题、类型和语法结构。文档和段落检索模块从文档集合中获得相应的可能,包括正确答案的段落集合。答案提取和验证模块根据前两个模块的输出及通过知识库和 Web 上的知识获得最终的答案。下面对这三个模块进行详细的介绍。

## 第二节 问句的分析

前面谈及问答系统和搜索引擎的不同时,着重强调了它们输入和输出的不一致性。问答系统的输入是由自然语言构造的问题,而搜索引擎面对的是构造好的关键词。如果把问题直接放入搜索引擎查询,大多数情况是难以获得满意的答案。因为搜索引擎一般不考虑问句的语法和语义的信息,它把整个问句切分成为一个关键词的集合进行查询。而且,很多疑问词和一些具有很强语义的虚词有可能被系统认为是停用词而过滤掉了。因此,对于自然语言问题,需要有特别的分析,以便于构造良好的查询并提取正确答案。

### 一、问句中的指代消解

人们提问,很多情况下并不是只有一个单一的问题,而是关于某一方面的一系列

问题。另外,当获得回答的时候,由于答案和期望的相互作用,用户可能转变了信息需求,产生新的问题。因此,问答系统应当是交互式的。在这样一个交互的环境中,问题就可能是上下文相关的了,它们里面可能会有有一些指代前边出现的实体的代词。考虑这样一个简单的情境:

用户:第一位登上太空的中国人是谁?

系统:杨立伟

用户:他在哪个单位工作?

在这里,如果只考虑单个问题“他在哪个单位工作”,那么系统肯定是无法找到正确的答案的,需要把“他”替换成“杨立伟”,这个问题变成“杨立伟在哪个单位工作”,系统才有可能找到答案。这个例子说明了指代消解在交互式问答系统中的作用。在TREC问答任务中,从2004年开始,它的主任务也不再是提出一个个独立问题,而是会根据一个主题问一系列的问题,那么在问题中某些代词其实也包含着某种指代关系。指代消解是自然语言处理中一个重要的问题,主要有两种解决方法:基于规则的方法和基于统计的方法,其中,基于规则的方法大多是根据解析获得的句法树过滤获得可能的先行语,并根据规则对候选先行语指定权值,取权值最高的先行语即为指代的对象。基于统计的机器学习方法可以把指代消解问题看做一个分类问题,它把当前的代词分到一个最合适的先行词中去。分类的特征一般包括:代词和先行词之间的距离,性别匹配度(当代词是指人的时候),单复数匹配度,语义匹配等,然后通过一些标注语料库的训练就可以得到一个分类的模型,从而对于新来的指代情况判断可能的先行词。

## 二、问句分类

问句分类是问题分析的重要组成部分,它直接对应在文档和段落检索中的检索方式及在答案抽取过程中选择什么样的答案。最初的问答系统也有不针对问题进行分类的,而是通过疑问词作为关键字决定是什么样的问题。在英语中,典型的有5W1H(What, Who, When, Where, Why, How)的问题,但是后来发现,这样的分类方法粒度太粗了,不能够满足问答系统的需要,特别是What, How其实可以涵盖许多种问题类,另外,有一些问句是祈使句,不包含疑问词,如“列出长江流经的省份”,或者有一些问句包含了多个疑问词,如英语中含有定语从句和宾语从句的情况,用这种方法就无法处理。于是有人希望能够对一些可能涵盖多种问句类型的疑问词建立规则进行细分,并且建立规则处理不包含疑问词或者含有多个疑问词的情况。更好的一种做法就是引入问句分类,所采用的大多是一些机器学习的方法。相应地,很多机构提出了一些分类体系结构,试图能够涵盖所有的问句,这些分类体系结构大多是层次化的,其中被引用最为广泛的是文献(Li et al. 2002)所给出的结构,如表13-1所示。这个结构针对问句的不同提问方式分为6个大类,分别是缩略语、描述、实体、人物、地点、数量。为了找到正确答案,在6个大类下面又分了50个小类,如在实体

类里面又有动物、颜色、创造者等,在数量类里面又有日期、距离、钱数等。问句分类的任务就是通过分类算法,把一个问句分到这样一个分类体系结构的一个或几个中去,由于某些问题比较模糊,可能属于一个以上的类别。

表 13-1 问题分类体系结构及 TREC 问答任务中问题的分布

Class	#	Class	#
ABBREVIATION	18	term	19
abbreviation	2	vehicle	7
expression	16	word	0
DESCRIPTION	153	HUMAN	171
definition	126	group	24
description	13	individual	140
manner	7	title	4
reason	7	description	3
ENTITY	174	LOCATION	195
animal	27	city	44
body	5	country	21
color	12	mountain	5
creative	14	other	114
currency	8	state	11
disease/medicine	3	NUMERIC	289
event	6	code	1
food	7	count	22
instrument	1	date	146
lang	3	distance	38
letter	0	money	9
other	19	order	0
plant	7	other	24
product	9	period	18
religion	1	percent	7
sport	3	speed	9
substance	20	temp	7
symbol	2	vol. size	4
technique	1	weight	4

问句分类可以看做是文本分类的一个延伸,但它和文本分类有很大的区别。一般文本都会有几千个字,而一个问句一般就几个字,这使得一些比较复杂的方法在文本分类中是难以实现的,但在问句分类中是可能的。也正是因为问句太短,所以仅仅从词这一个级别无法获得足够的信息特征,需要提取一些更深层次的信息作为分类的补充特征。

对于一个问句,可以提取的特征有很多。最简单且最常用的是用关键词包(bag

of terms)的方法,把所有出现的关键词汇作为特征向量。这种方法,在信息检索(IR)和文本分类(TC)中应用的很普遍。而对于绝大部分问句来说,由于它们本身的长度很短,因此一个词汇基本上都只能出现一次,所以这是一个0,1组成的向量。但是,这样会丢失很重要的词汇出现序列的信息。在自然语言处理(NLP)领域,解决这个问题的方法是引入 $n$ -gram模型,即考虑 $n$ 个连续的词汇序列出现的计数。其实关键词包就是1-gram(unigram)的模型。一般来说,高维 $n$ -gram的特征会和低维gram特征向量一起使用。同时, $n$ 值也不能取得太大,因为 $n$ 越大,数据稀疏问题就越严重,一般取值是2~3。以上的方法是最简单的两种,只使用了表层的语言特征,更进一步的,一些隐含的语法层面的特征对问句分类也很有帮助。对词汇进行词性的识别(Part of Speech)也可以形成由词性组成的向量构成特征。更高级的,把问题进行语法分析,构成句法树,不同问题之间的距离可以用衡量句法树相似度的一些指标如 kernel tree similarity 来计算。也可以利用语义的信息,如可以利用 WordNet 或者 HowNet 的信息构造由同义词/上层词构成的向量来表示这些特征。

很多通用的机器学习算法都可以被用到问句分类中,如 $K$ 最近邻算法、朴素贝叶斯算法、决策树、神经网络、支持向量机(SVM)算法等,其中,文献(Zhang et al. 2003)显示,如果只采用表面的信息( $n$ -gram),用SVM进行问句分类会获得好的分类结果。

此外,某些系统的问题分析还要进行主题及关键词的提取、词性标注、或者句法分析等。

### 三、问句主题提取

问句分析的另一个方面是问句主题提取。在后续的检索模块中,需要选择问题中的一些关键词进行查询,必要的时候会对查询进行调整,但都应该包含这个问题的主题。通常可以通过对问句进行句法分析,获得这个问句的中心词,然后选取中心词和相关的修饰词作为问题的主题。如何选取合适粒度的中心词组是问题的关键。文献(Cui et al. 2004b)提出了一种基于外部资源选取词组的方法。它把问句中的关键词提交给搜索引擎,从搜索引擎返回的答案中发现各种词的组合的点互信息,只有点互信息高于一定程度的中心词的组合才被认为是词组,这个词序列就构成了问题的主题。

在一些系统的问题分析中还包含产生查询关键词,但由于关键词提取会依赖于检索模块所采用的模型,而且有的提取算法需要和检索模块的迭代过程从而和检索算法高度耦合,因此可以把这个子模块放在检索模块中介绍。

## 第三节 文档和段落检索

这个模块可以看做是一个信息检索子系统。有一些问答系统直接使用已有的检索系统(如 Smart, Lemur, Lucene 等)或者搜索引擎(如 Google)对问题的非停用词进行

检索,把获得的结果直接作为答案提取和验证模块的输入之一。这是一种最简单的方法,但往往会导致比较低的查准率和查全率。文献(Collins et al. 2004)通过实验验证了文档检索的好坏会直接影响问答系统的整体性能。当一个检索系统的查准率比较差的时候,可能会有大量的无关文档需要后续的模块处理,而一般来说,答案提取和验证模块需要比较复杂的自然语言处理的技术,因此大规模的无关文档会大大降低系统的效率。如果检索系统的查全率比较低,那么也就意味着有很多包含答案的文档或者段落没有被检索到。包括正确答案的文档或者段落越少,那么提取出正确答案的可能性也越小。在极端的情况下,如果所有包含答案的文档都没有检索获得,那么后续的模块也无法获得答案,很多研究者认为在这一阶段,查全率比查准率更为关键。所以,需要专门为问答系统设计一套检索系统,从而希望能够获得更好的查准率和查全率。

### 一、检索模型的选用

信息检索的经典模型包括布尔模型、向量空间模型和概率模型等。其中布尔模型是最简单的一种,它把关键词和关键词组用一个布尔表达式表示,使得文档中出现的关键词满足这个布尔表达式。它的优点在于简单和高效,但是它没有提供对文档和段落进行排序的功能。所以一些搜索引擎会提供布尔查询的界面,即允许用户输入 AND, OR, NOT, 括号等,先通过布尔查询获得候选文档集,然后对这些文档用其他模型进行排序。向量空间模型的基本思想是每一个词对应向量中的一项,通过文档/段落中出现的频率(TF)和文档集中出现频率(DF)计算该项的权值,比较查询和文档向量的相似度(一般用余弦夹角值)得到文档相关度,从而选取相关性比较高的文档集合作为查询的结果。概率模型是在已知查询的情况下,计算文档满足这个查询的概率有多大,即计算  $P(\text{Doc} | \text{Query})$ , 概率比较大的文档组成的集合作为查询的结果,在评测中基于概率模型的 BM25 算法的效果是名列前茅的。文献(Moldovan et al. 2003, Tellex et al. 2003)通过实验发现在问答系统的文档检索中,简单的布尔模型的效果和概率模型及改进了的向量空间模型不相上下,因此我们认为通过布尔模型就可以满足问答系统的文档检索需求。

### 二、查询生成

无论采用何种模型,检索系统的输入应该不是一个问句,而是由关键词组和关键词组成的查询。因此这里所需要做的第一步,就是把问题转换成查询。

最简单的转换方法就是把问句中的停用词去掉,其余的词作为关键词进行检索,也有一些系统采用这样的方法,但使用该方法存在几个问题。

首先,问句是可长可短的,长的问句可能有很多字,譬如“毛巾上无缘无故出现一些蓝色的、黑色的不明物质,又好像是化学反应,到底怎么回事?”。短的问句也就只有寥寥几个字,譬如“谁是孙立人?”,只有一个关键词,即“孙立人”。当问题很长的时



候,这样生成的关键词就会很多,如果采用布尔模型进行检索的话,可能没有任何文档或者段落包括这么多的关键词。当问题很短的时候,关键词只有一两个,那么包含这些关键词的文档又会太多,不利于后期的处理。

相比于信息检索获取相关性文档,一个问句在一个文档集中存在特定答案的可能性会更小一些,在很多情况下,虽然包含了答案可以回答一个问题,但由于问题采用了和文档不同的词或者不同的形态导致找不到答案。譬如问句是“桑塔纳 3000 是哪个公司生产的?”,在文档集里面有这样的文本“由上海大众自主开发设计制造的桑塔纳 3000 自年初上市以来,销量已经突破……”,虽然很明显这段文本中包含了回答问句的信息,即“上海大众”,但是它却没有包含“生产”这个关键词,而只有“设计”,“制造”这样意义相近的词。

基于以上考虑,如果简单地停用词直接搜索效果不会很好。为了解决第一个问题,我们需要对关键词进行调整,如果关键词太多了,从而查询条件过于紧,应该去掉一些关键词。如果关键词太少,查询过于宽松,就应该加上一些关键词。对于第二个问题,自然的想法就是把词汇的词形变化和意义接近的词汇加到查询中来。

我们提出了一种迭代的选择关键词和查询扩展的方法,并在 TREC2004 和 TREC2005 的数据上做了实验,获得了比较好的结果。在这种方法中,选用布尔模型作为检索模型,首先需要从问题中选择关键词。名词和命名实体往往是问句当中包含信息量较大的部分,因此我们选择它们并且进行合取操作组成初始查询。我们需要设定一个合理的返回文档数区间,只有当返回的文档数在这个区间内的时候,才称这个查询合适。如果查询过松,即能够得到的文档数超过了这个区间的最大值,我们需要收紧查询。最简单的收紧查询的方法是增加合取项,合取项的内容是问句中的关键词包括动词,形容词和副词,加入时可以按照包含信息度的多少从大到小的顺序,一个词包含的信息度用它的 IDF 值表示。另一种收紧查询的方法是取消词汇的扩展形式,即在查询中只考虑原问句中的词汇,而去掉它的词形和语义上的扩展。如果查询过紧,即能够得到的文档数尚不到这个合法区间的最小值,需要放宽查询。放宽查询可以加上词汇的词形扩展和语义扩展作为析取项。词形扩展在一些具有词的曲折表示的语言中特别常见,如英语中对于一个词可以加上例如“ing”,“ed”,“s”之类的后缀表示单复数或者时态,主被动情形下的变化作为查询。语义扩展就是加上一些意义相同或者相近的词汇作为查询,通常会利用一些机器可读的词典如 WordNet 等。放宽查询还可以减少布尔查询的合取项。另外一个放宽查询的方法是把作为词组查询的命名实体拆分,如在问题中有“北京大学网络和分布式实验室”这一实体名,当发现加上该实体后只能找到很少的结果时,可以拆分成“北京”,“大学”,“网络”,“分布式”,“实验室”这一系列关键词,这样就可以获得更多的结果。根据这些基本的放宽和收紧查询的方法,使系统不断迭代,使得对多数问题的查询结果中所包含的文档数在这个合理的区间内,从而获得了查准率和查全率的折中。

上述方法的查询关键词是从问句出发的,采用了一些选取的策略和基于词典扩

展的方法。另外一种构成查询的方法是从答案出发的,对于某一种问题,系统从大量的训练数据中学习对这一问题的回答模式,根据这个模式对问句进行重写,构造查询。文献(Agichtein et al. 2004)就是采用这种方法来做的,这种方法通常需要一个比较大的问题-答案对的训练集,而这个训练集的来源大多是一个常见问题解答库(FAQ)。首先,需要从大量的问句中统计出问句词组,也就是经常作为问题的词组,通常采用  $n$ -gram 统计的办法。然后,对于每一个问句词组,都要对应一些问题-答案对,同样的,对于在答案文本中的  $n$ -gram,统计它们的出现频率,对于出现频繁可能构成模式的  $n$ -gram,称之为答案词组,就作为一种候选重写规则。为了防止和内容相关的词组被错误地认为是模式包含到重写规则中来,需要一个对答案词组进行过滤的步骤。一条比较明显的过滤规则就是如果某个答案词组中出现了名词,那么它很有可能和内容相关,就认为它不是模式。对于已经找出来的候选模式,需要对其进行排序,以获得更加有效的模式。对于候选模式的打分方式借鉴了 Okapi BM25 的 词项权重公式及考虑答案词组出现的频率,其定义为

$$wtr_i = qtf \cdot w_i$$

$$w_i = \frac{(r + 0.5)/(R - r + 0.5)}{(n - r + 0.5)/(N - R - r + n + 0.5)}$$

其中,  $qtf$  表示该  $n$ -gram 和问句词组的共现次数,  $r$  表示包含某个答案词组的相关文档数,  $R$  表示总相关文档数,  $n$  表示包含了某个答案词组的文档数(不考虑该文档相关与否),  $N$  表示总文档数。

这里和 BM25 在信息检索环境中使用的区别在于,在信息检索中相关或者不相关文档都是针对一个具体的查询而言的,而在这里,相关和不相关是针对某一个特定的问句词组的答案。这样已经得到了一系列备选的重写规则和它们的权重。但是对一个常见问题与答案集合,数据往往不够大,获得的权重有可能不准确,因此还可以通过搜索引擎进一步验证这些重写规则。

在获得了一些重写规则后,系统就可以根据这些重写规则构造查询,如有一个问句词组是“*What is a*”,对应的重写规则包括“*is used to*”,“*according to the*”,“*to use a*”,“*is a*”,“*of a*”,“*refers to*”,“*used*”,“*refers*”,“*usually*”等。当有一个问句“*What is a computer*”的时候,就可以构造查询“*computer AND ‘is used to’*”,“*computer AND ‘according to the’*”等,就有可能找到正确的答案。这种方法的好处在于,它在构造查询的时候就预先考虑了答案可能包含的与内容无关且只与问题形式有关的关键词,但是这种根据问句词组到答案词组的映射是表层的,可能存在相同的问句词组在上下文中会有不同语义的情况,映射成的答案词组可能是错误的。另外,这种基于统计的方法获得的映射也是不完备的。

### 三、查询结果排序

通过上述步骤,已经可以从问句中构造出查询,并且通过布尔检索模型获得一个

相关的文档集了。在问答系统中,考虑到后续步骤的计算复杂性,检索系统的结果应该是段落的集合,这种段落可以是固定字节数的文本,或者  $m$  个句子,也可以是原文本中的一个自然段落。在选择段落的时候,需要有一种对段落相关性判断的方法,并且可以对段落进行排序输出,下面讨论解决这一问题的几种方法。

最简单的方法是根据文档的自然段落,采用一种信息检索的模型(如 BM25),把自然段落看成一个文档,对所有的自然段落打分排序。文献(Tellex et al. 2003)通过实验的方法考察了各种段落检索的算法。实验结果表明,基于密度的算法可以获得比较好的效果。该算法是由查询关键词在某个段落里的出现次数和邻近程度来决定这个段落的相关程度。几种表现比较好的段落检索算法如下。

MultiText 算法:这种算法会倾向于挑选比较短的且包含尽可能多的高信息量(对应于比较大的 IDF 值)关键词的段落。该算法会检索出文档中查询关键词密集出现的段落,这种段落的定义是从一个查询关键词开始,到一个查询关键词结束,且中间包含了尽可能多的查询关键词。因此,这种段落是不定长的,这里定义相关性和段落的长度成反比,因此倾向于获得更短的段落。

IBM 的算法:这是 IBM 在参与 TREC 评测中提出的算法。它提取了一些相关性的特征,包括:匹配的关键词特征,是指同时在查询和段落文本中出现的关键词的 IDF 值的和;词典匹配关键词特征,是在查询中的关键词,虽然没有在段落中出现,但其同义词出现了,这些关键词的 IDF 值之和;不匹配关键词特征,是虽然在查询中出现了,但是在段落中没有出现的关键词的 IDF 值之和;分散程度特征,是在匹配的关键词之间的间隔;聚类词特征,即同时在问题和段落中都邻近出现的词的数目。最后通过线形叠加累积这些特征对于相关度的影响。

SiteQ 算法:该算法规定检索获得段落是由  $m$  个句子构成的。对于每个句子,获得的分数由两部分组成,一部分是所有关键词的 IDF 值的和,另一部分是相邻关键词的距离的平方倒数和 IDF 和的乘积。段落的得分是  $m$  个句子得分的叠加。

考察上述三个算法,虽然在设计和实现细节上有很大的差异,但是都使用了 IDF 值的和以及引入了邻近关键词之间的距离。所以,在进行问答式系统的段落检索时,基于密度的算法是有效的。

上面的算法只考虑了独立的关键词的信息,没有考虑关键词在问题中的先后顺序,也没有考虑语法和语义的信息。为考虑语法信息,最直观的想法就是把问句和答案都解析成语法树,从两者语法树的结构中找出一些相关性的信息。文献(Cui et al. 2005)提出了一种基于模糊依赖关系匹配的算法。这种算法需要把问题和答案都解析成为语法树,并且从中提取词与词的依赖关系。由于同样的问句可能具有语义上相同但是语言表述上不同的回答形式,所以如果只考虑相同类型的依赖关系会导致比较低的查全率,因此需要度量依赖关系之间的匹配程度。在训练数据上,通过统计依赖关系发现,可以获得依赖关系的匹配程度。在检索排序的时候,通过依赖关系匹配度,就可以获得问句和候选答案句子在语法上的匹配度。实验结果表明,这种方法检索的效

果比基于密度算法的 SiteQ 为优。

以上介绍了文档和段落检索的一些方法,概括起来说,就是从问句中挑选合适的关键词作为查询,进行一定的扩展,加入一些答案词组作为附加关键词。查询获得文档集,排序获得相关性段落,且检索中最好能够使用关键词距离的信息,即基于密度的算法,另外加入一些语法的信息也有利于检索效果的提高。

#### 四、增强索引的功能

大多数的问答系统都基于传统信息检索系统的全文索引。但是,和一般信息检索系统只需要处理关键词不同的是,问答系统需要处理更多的语法、语义信息。因此,有一些系统把这些信息也放入到索引中去,以提高效率。文献(Radev et al. 2000)把一些关键词或者词组的属性放入索引,在组成布尔查询的时候,不仅需要包含某些关键词,还有答案或者关键词的属性要求。例如,对于一个问时间的问题,就要求返回的段落中包含时间。文献(Bilotti et al. 2007)把查询变成一个结构化的查询,表达查询词和段落中应该包含的某些词的属性。进一步的,还有一些系统索引了句子中不同的词或者词组的关系,文献(Katz et al. 2003)索引了段落中句子解析称三元组的形式,索引这样的三元组。文献(Chu C et al. 2006)还索引了句子中词和词组的语义关系。

### 第四节 答案提取和验证模块

一个问答系统通过问句分析和文档段落检索模块可以获得问题答案的段落集合,答案提取和验证模块从这些段落中获取正确的答案。为了提取答案,一般首先需要通过精细实体识别,从段落中识别出符合问题类型的答案作为候选集合,然后利用表层字符串特征,或者语法、语义、逻辑上的关系,或者模式匹配来获取答案。通常,获得答案还可以利用一些外部的资源(如 Web、WordNet、知识库等)进行验证,对结果进行重排序。

#### 一、生成候选答案集合

问题分析模块已经获得问句分类。观察表 13-1 中的问题类型可以发现,除了描述类(Description)和其他几个小类(如 expression、order 等),大多数的问题类型对应的答案都会比较短,它们可能是实体名(如人名、机构等),也可能是属于一定语义范畴的名词(如食物、技术、树木、花朵),也可能是数字度量(如距离、速度等)。对于这些类型的问题,可以通过找出相应类型的词、词组或者片断来回答。对于传统的命名实体(如人名、机构、地点、时间等),在自然语言处理领域已经有比较深入的研究,只要通过一个训练获得模型(如隐马尔科夫模型(HMM)或者条件随机场模型(CRF))就可以从文本中识别。对于一定语义范畴的名词(如鸟类、花朵等),可能需

要事先搜集获得属于该语义范畴的具体名词集合,只要在段落中寻找这些集合中的词作为候选答案即可。为了构建具体名词集合,可以借助机器可读的词典如 WordNet 等。但是 WordNet 主要是一个概念词典,包含的具体名词会很少,不能够满足回答这类问题的需求。为了获取更多的具体名词,一种方法是通过一些〈概念性名词,具体名词〉作为训练的种子,从大的文档集或者 Web 中找到这种连接概念性名词和具体名词的模式,再根据这种模式提取更多的具体名词,多次迭代可以发现更多的〈概念名词,具体名词〉对和相应的模式。这种方法称为自举(Bootstrap)方法。文献(Mann 2002)就是采用这样的方法,在 WordNet 的概念名词的本体结构之下再建立一层具体名词的本体结构,这层具体名词结构有助于从检索的段落里获得语义上匹配的短语。另一种简单的方法是自动地提取 Web 资源(如 Wikipedia, CIA 等)中具体名词列表。对于可以用正则表达式表示的答案,可以手工定制一些正则表达式包含尽可能多的情况,如距离一般可能是一个数字后面跟上一个表示距离的单位。上述识别方法,能够覆盖大多数问题类型。通过在文本中匹配相应类型的短语,即可构成候选答案集。

## 二、答案提取

在获得候选答案集合以后,如何获得匹配问题的最佳答案呢?首先介绍 3 种可以利用的不同层次的特征及相应的方法,最后介绍建立针对答案提取的统计学模型。

### 1. 基于表层特征的答案提取

早期的大多数系统只使用表层特征来抽取答案。一类表层特征是答案周围段落的一些特征,如段落和查询的相关程度,所有查询词之间的距离,查询词和候选答案的距离。一般来说,段落相关程度过高,查询词之间及查询词和候选答案之间距离越接近,则该候选答案越可能是问题的答案。另一个特征是该候选答案出现的次数。对于一个比较大的文档集,一个问题的答案可能反复出现,出现的次数越多,则它越可能是正确答案(Lin et al. 2003)。同理,该问题的答案可能在更大的数据集(如 Web)中反复出现,因此可以通过搜索更大数据集提取候选答案,从而根据总体的重复度估计它的正确性,这是一种广泛采用的答案验证方法。可以通过一个模型综合这两类特征,获得候选答案的得分。

### 2. 通过关系抽取答案

表层特征没有考虑语法、语义的因素,因此很有可能出错。文献(Light et al. 2001)估计这种基于实体识别和表层特征的方法的性能上限是 70%。如对于两个不同的问题“青蛙吃什么动物?”和“哪些动物吃青蛙?”通过问句分析,它们对应的问句类型都是生物。通过文档和段落检索,由于查询关键词均为“青蛙”,“吃”。检索出来的文档包括这样的一些句子:“成年的青蛙主要吃昆虫,偶尔也会吃小鱼、蚯蚓和蜘

蛛”，“鳄鱼一般会吃水边的动物，如鱼、蛇、青蛙、乌龟和一些哺乳动物”。因此，一个基于表层特征的系统，是无法区分这两个问题的。这类方法具有局限性，为了克服上述缺陷，不少研究者提出了不同的改进。START 系统采用的办法是把问句和文本中的句子转换成三元组的形式，三元组基本的构成是〈主语，动词，宾语〉，去掉了句子中的一些修饰成分，如上面场景中的问句变成的三元组就是〈青蛙，吃，什么〉和〈什么，吃，青蛙〉，与第一个问句相匹配的文本三元组应该是〈青蛙，吃，昆虫〉，〈青蛙，吃，小鱼〉等，与第二个问句相匹配的文本三元组应该是〈鳄鱼，吃，青蛙〉、〈蝙蝠，吃，青蛙〉，这样就容易从文本三元组中获得答案。

另一种解决方法是采用逻辑建立问题到答案的关系(Moldovan et al. 2002)，在历届 TREC 问答系统评测中表现最好的语言计算公司(LCC)多数采用这样的方法。逻辑表示是介于句法解析表示和深层语义表示之间的一种表示形式，它可以通过解析获得的句法树及一些规则计算获得，表达了主语、宾语、前置词、复杂的名词性短语、附属的形容词或副词等。由于自然语言的复杂性，存在非常多这样的转换规则，且有 10-90 的规律，即可以通过 10 条最频繁使用的规则覆盖 90% 的情况。转换成逻辑表示以后，通过一个逻辑证明机，借助一些外界的知识，如果可以通过答案来证明问题，那么这个答案就是正确答案。

### 3. 通过模式匹配抽取答案

基于句子级别的模式匹配来进行答案抽取，该模式可以通过手工定制或机器学习获得。一般来说，手工定制模式扩展性和覆盖率都比较低，主流的方法是基于训练数据的机器学习方法。

文献(Cui et al. 2004a)提出了一种软模式的方法处理定义类的问题。它首先对答案语句进行词性标注(POS)，由于模式只是一种回答的方式，而需要尽量少的训练集的内容，因此需要替换包含内容语义的词汇为它的词性。以查询关键词为中心取一个窗口，设窗口的长度为  $2w+1$ ，其中在关键词前面有  $w$  个词，在关键词后面也有  $w$  个词： $\langle \text{slot}_{-w}, \text{slot}_{-w+1}, \dots, \text{slot}_{-1}, \text{查询关键词}, \text{slot}_1, \dots, \text{slot}_w \rangle$ 。

考虑在该窗口中词出现的两种规律，即绝对位置和相对位置的分布规律。我们称窗口中的第  $i$  个位置为第  $i$  个槽。统计绝对位置的信息，就是考虑第  $i$  个槽的词汇出现的可能性，即  $\Pr(\text{token}_i | \text{slot}_i)$ 。统计相对位置的信息，就是考虑前后相邻词的相互依赖关系，统计 bigram 的情况，即  $\Pr(\text{token}_i | \text{token}_{i-1})$ 。

$$\Pr(\text{token}_i | \text{slot}_i) = \frac{f(\text{token}_i)}{\sum_{\text{token}_i \in \text{slot}_i} f(\text{token}_i)}$$

$$\Pr(\text{token}_i | \text{token}_{i-1}) = \frac{f(\langle \text{token}_{i-1} \text{token}_i \rangle)}{f(\text{token}_{i-1})}$$

其中， $f(\text{token}_i)$  为该词在第  $i$  个位置出现的计数， $f(\langle \text{token}_{i-1} \text{token}_i \rangle)$  为  $\text{token}_{i-1}$  和  $\text{token}_i$  连续出现的计数， $f(\text{token}_{i-1})$  为  $\text{token}_{i-1}$  单个出现的计数。通过这些信息，

就可以计算候选答案语句和模式的匹配程度。分别计算绝对位置的匹配程度  $Pa\_weight_{slots}$  和相对位置的匹配程度  $Pa\_weight_{seq}$ , 综合获得最后的总匹配程度。

$$Pattern\_weight = \frac{Pa\_weight_{slots} \times Pa\_weight_{seq}}{fragment\_length}$$

$$Pa\_weight_{slots} = Pr(S | Pa) = \prod_{i=-w}^w Pr(token_i | slot_i)$$

$$Pa\_weight_{seq} = (1 - \alpha)Pr(left\_seq | Pa) + \alpha Pr(right\_seq | Pa)$$

$$Pr(left\_seq | Pa) = Pr(token_{-w}, token_{-w+1}, \dots, token_{-1} | Pa)$$

$$= Pr(token_{-w})Pr(token_{-w+1} | token_{-w}) \dots Pr(token_{-1} | token_{-2})$$

$$Pr(right\_seq | Pa) = Pr(token_1, token_2, \dots, token_w | Pa)$$

$$= Pr(token_1)Pr(token_2 | token_1) \dots Pr(token_w | token_{w-1})$$

其中,  $Pattern\_weight$  表示候选答案语句和模式的匹配程度,  $fragment\_length$  所取得窗口的大小,  $Pr(left\_seq | Pa)$  和  $Pr(right\_seq | Pa)$  分别代表左半部分相对位置匹配程度和右半部分相对位置匹配程度。通过以上的方法获得的模式匹配度和内容匹配度相结合, 就可以获得相关性, 且选择相关性程度比较高的句子作为候选答案。

#### 4. 答案提取中的统计模型

在问答系统的研究中, 有学者用统计模型对答案提取模块进行建模。如下介绍两个有代表性的建模方法。

1) 噪音信道模型(Echihabi et al. 2003)。该模型把问句看成是目标信息, 把答案看成源信息, 假设源信息需要通过一个包含噪音的信道, 需要估计  $Pr(q | S_{i,a})$ , 也就是一个包含候选答案的句子  $S_{i,a}$  变成问句  $q$  的概率。这里的问句和答案句子都表示成关键字及其类型(词性)的序列, 把噪音建模成一组随机操作, 如添加、删除、替换、对齐词或词组等。该模型通过一组随机操作把  $S_{i,a}$  变成  $q$  的可能性近似为一个包含答案的句子能回答问题的可能性。各种操作发生的概率可以通过一组训练数据(问题-答案对集合)训练而获得。

2) 采用无向图模型对它建模(Ko et al. 2007)。图模型是近年机器学习领域非常热门的一个模型族。该模型同时考虑答案和问题的相关性及答案之间的相关性。对于一个问题的一组答案  $\{A_1, \dots, A_n\}$ , 用一组二元变量  $S = \{S_1, \dots, S_n\}$  来表示每个答案是否为正确答案, 若  $S_i = \text{TRUE}$ , 则  $A_i$  就是正确答案, 反之, 若  $S_i = \text{FALSE}$ , 则  $A_i$  就不是正确答案。  $Pr(S)$  表示用  $S$  的取值来判定答案正确与否的可能性, 因此  $S_0 = \arg\max Pr(S)$  就是一种判断答案正确与否的最佳方式。  $Pr(S)$  由答案和问题的匹配程度及答案之间的相关程度共同决定, 即

$$Pr(S) = \frac{1}{Z} \exp(\sum_{i,j} \theta_{ij} S_i S_j + \sum_i \theta_i S_i)$$

其中,  $\theta_{ij}$  表示答案  $A_i$  和  $A_j$  的相关性,  $\theta_i$  表示答案  $A_i$  与问题的相关性, 而这里答案之

间或者问题答案的相关性都是由一组特征线性加权获得的,各个特征的权重是通过训练数据而获得的。

## 第五节 问答系统的改进方法

本节讨论一些改进方法,有的系统利用一些其他的外部资源来提高系统的效果,有的特别关注解决某种类型的问题,有的致力于构建大量的训练数据,也有的采用系综学习(Ensemble Learning)的办法做系统级别的合成。

### 一、问答系统中外部资源的利用

在传统方法中,已经大量地采用了外部资源。例如,在问句到查询的转换中,可以用 WordNet 做了查询扩展;在答案的提取和验证中,可以用 Web 数据进行答案的验证。下面讨论这些外部资源的进一步利用,以及其他可用的外部资源。

在问句分类中介绍了一种分类体系结构。但是,也有的研究者认为,这种结构有点不伦不类,其中有的类别表示的是一种语义上的信息,而其他的一些类别表示的是结构形式上的信息。文献(Pasca et al. 2001)把答案类型按在 WordNet 中的语义类来进行划分,在答案提取的时候,要求答案与其类型相匹配。

在 Web 上,存在着大量的整理好的结构化数据,这些数据一般通过表格或者模板类网页的形式展现出来。由于它们具有比较固定的格式,比较容易从它们的格式中理解语义,因此这些数据可以事先抓取下来作为本地的知识库。这些知识库可以组织成为三元组<实体,属性,值>的形式,可以通过分析问题的语法结构,明确问题是不是也能重写成三元组。如果可以,则做一些语义上的扩展并且从知识库中查询,就有可能获得正确的答案。由于这些数据来源比较权威,语义信息明确,因此可以在回答中赋予比较高的置信度。

在回答定义类的问题时,除了以上所说的采用模式的方法以外,也有很多外部资源可供利用。例如,在 WordNet 中,很多词汇已经有了非常好的定义可以直接使用,而且词在这个语义网络中所处的拓扑结构位置也包含了它的定义信息。在 Web 上存在一些类似于百科全书的网站,如 Wikipedia 等,利用这些资源可以获得定义。另外,把查询关键词输入搜索引擎,也可以大致地获得这个关键词的一些定义信息。文献(Cui et al. 2004c)比较了这三种方法对回答定义类问题的效果,比较的结果是百科类网站结果优于 WordNet 的结果,而 WordNet 的结果优于搜索引擎返回的结果。显然,这是由于前两者都是花费大量的人工整理的,准确性比较高。而与 WordNet 相比,百科类网站包括了一些具体事物的定义,信息比较全面,回答的质量要好很多。

还有一类很重要的资源是常见问题解答,这类资源已经很丰富了,如常见问题档案馆(<http://www.faqs.org>)就提供了大量的资源。另外,某些社区里面存在大量



的常识性问答,还有专门的这类问答式的社区,如 Wondir(<http://www.wondir.com>)和百度知道(<http://zhidao.baidu.com>)。在这些社区中,注册用户每天会提出很多问题,其他的注册用户回答这些问题,通过一定的投票验证机制确定最佳答案,并予以回答问题的用户积分和头衔表示激励。所以这些社区里面也有很多这样的问题-答案对资源。用户可以利用这样的一些资源帮助自动问答系统提高效果。这时候,不再像上面那样通过问句分析、段落检索、答案提取这三个步骤获取答案了。而只需要比较新的问句和常见问题解答库中问题-答案对的相似度,找出和问句所问内容相同的已知问句,用它的答案直接回答。这样做的好处是:比起传统的问答系统,答案是经过人工编辑和验证的,更加详尽,更有意义。比起问答社区,它不需要几天甚至几十天的等待,可以实时的获得答案。

如何找到匹配的问题答案对呢?首先想到的是比较问句和问句的相似度,但是问句都很短,而且可能同样语义的问题会采用不同的词汇,不同的句式表达出来,因此仅仅比较问句相似度可能会有很大的误差。而在已回答的问题中,答案会提供比问句更多的信息,如果把整个问题-答案对统一起来考虑,很可能会得到更好的结果。文献(He et al. 2005, Jijkoun et al. 2005)都采用了按照文档,问题-答案对和问题多层相似度叠加的方法,获得了比较好的性能。

在很多问答社区里面,由于很多用户并不习惯于搜索以前问过的问题,因此,同一语义的问题有时候是重复存在的,因此有很多冗余问题存在。可以利用这些冗余的问题-答案对,发现词义之间的相关性。文献(Jeon et al. 2005)利用机器翻译模型计算将一个问题翻译成另一个问题的可能性,从而找出最可能的同义问题。这个方法首先是通过比较答案的相似度,从问答库中找出可能同义的问题-答案对,并把这些问题-答案对作为训练集训练获得翻译模型。翻译模型的基本原理是对于两个词,计算它们的转移概率。

$$P(t | s) = \lambda^{-1} \sum_{i=1}^N c(t | s, J^i)$$

$$c(t | s, J^i) = \frac{P(t | s)}{P(t | s_1) + P(t | s_2) + \dots + P(t | s_n)} \#(t, J^i) \#(s, J^i)$$

其中,  $t$  和  $s$  表示两个词,  $P(t | s)$  表示从  $s$  转移到  $t$  的概率,  $\lambda$  是一个归一化因子,  $J^i$  表示一对相同语义的问题-答案对,  $c(t | s, J^i)$  表示在  $J^i$  中  $s$  转换为  $t$  的可能性,  $\{s_1, s_2, \dots, s_n\}$  表示出现在  $J^i$  中的词汇的集合,  $\#(t, J^i)$  和  $\#(s, J^i)$  分别表示  $t$  和  $s$  在  $J^i$  中出现的计数。可见,这是一个迭代的过程,初始化的转移概率是随机数,可以证明,这个过程是收敛的,最后可以获得恒定的转移概率。在通过训练获得词与词的转移概率以后,就在一定程度上得到了词和词之间的语义相似度,从而可以计算新的问句和已知问题-答案对的相似度了。相似度的计算方法如下。

$$\text{sim}(Q, D) = P(Q | D) = \prod_{w \in Q} P(w | D)$$

$$P(w | D) = (1 - \lambda) P_m(w | D) + \lambda P_m(w | C)$$

$$P_{ml}(w|D) = \sum_{t \in D} P(w|t)P_{ml}(t|D)$$

其中,  $\text{sim}(Q, D)$  表示问题和已知问题-答案对的相似度,  $P(w|D)$  是由已知问题-答案对推出某一个词的可能性, 为了防止某些词在训练集中没有和问题-答案对里的词形成相关性而导致总相关性为零,  $\lambda P_{ml}(w|C)$  起到了平滑的作用,  $P_{ml}(w|D)$  的表达式中就用到了训练集中计算获得的转移概率。

这种基于翻译模型的方法考虑词与词之间语义的相关性, 允许使用不同的词表示相同的语义, 因此效果会比简单的直接检索关键词好。实验证明, 这样做的效果比语言模型、OkapiBM25 和空间向量模型都好。

## 二、寻找特殊类问题的解决方案

分析理解 TREC 中间答任务的结果可以发现: 各种不同类型问题的回答效果差别很大, 如实体类的问题、回答的效果很好, 但是描述类问题、原因类问题的效果并不好。因此, 某些研究致力于某些特殊类问题的解决, 它们一般是在传统的处理流程之外加上几个特殊的问题处理模块, 当发现问题类型和处理模块可以处理的类型相匹配的时候, 就用该模块处理。主要研究的问题类型包括, “谁”问题(Who), 原因问题(Why), 方式问题(How)等。

问“谁”的问题其实有两种截然相反的类型, 第一种典型例子是“谁是北京大学校长”, 这种问题的已知是一个概念, 未知的是这个概念的一个实例; 第二种的典型例子是“谁是周其凤”, 这种问题的已知是一个实例, 未知是这个事例所隶属的一个概念。显然, 如果能够从文本中预先提取出概念-实例对, 则可以帮助回答这类问题。在文献(Fleischman et al. 2003)中, 为了提取这种概念-实例对, 考虑了英语中的两种语言现象, 一种是在一个概念后面直接跟上一个实例, 如“president Lincoln”, 这种现象在中文中也同样存在, 如“北京大学校长周其凤”; 另一种语言现象是同位语, 如“Bill Gates, CEO of Microsoft”。该文通过构造两个正则表达式涵盖了这两种语言现象。但是, 这里还存在一个准确率的问题, 即并不是所有能够和这种模式匹配的语言现象一定是人物的实例和概念, 为了提高准确率, 该文又提取了一些特征, 如模式的可信度, 概念词的结尾, 概念词的类型, 实例词的特征等, 标注了一些满足模式文本属于或者不属于概念-实例对, 通过机器学习的方法训练出一个分类器, 从而对满足模式的文本进一步分类。这样, 就可以在问题到来之前把文本中所有的概念-实例对都提出来组织成知识库的形式, 当遇到这类问题的时候, 跳过问答系统的传统解决过程, 直接到知识库里查找。

原因类问题也是颇具难度的一类问题。在很多情况下, 原因和结果是通过动词相连接的, 如 cause, lead to, bring about 等是最常见也是最明显的包含因果关系的动词, 还有一些动词在语义中已经包含了结果, 如 kill 中包含 death 的结果, dry 包含 dryness 的结果等, 一些动词甚至包含着事件的内容, 如 poison 表明结果是 death, 手

段是 with poisoning。文献(Girju 2003)希望找出表示一定因果关系的动词及它们的模式。该文首先借助 WordNet 寻找一个具有因果关系的名词对,方法是在 WordNet 的一个名词 A 的定义中如果出现“A cause B”或者“B caused by A”这样的短语,那么认为 A 和 B 具有因果关系,按照这种方法就可以收集一个具有因果关系的词汇集合。然后,通过这个集合可能找到包含了一定因果关系的动词,方法是在文档集中检索具有因果关系的词汇对,并连接它们的动词进行排序,获得出现频率比较高的动词作为候选动词。由于在语言中往往有搭配的概念,某个词和某些词在一起出现表达了因果关系的意义,而和另外一些词出现可能会表示另外的含义。为了区别不同的名词,该文依旧采用 WordNet 中的词的分类,把名词分为 9 种类型:实体、心理活动、抽象名词、状态、事件、动作、组织、过程和现象。这样,对于三元组(原因,动词,结果),每一项都会有一个取值。通过标注一些训练数据,就可以训练一个分类器(该文使用的是决策树算法 C4.5),根据取值就可以知道在某个情境下,是否表示一个因果关系。对于是因果关系的文本,就可用于回答因果类的问题。

问方式或者性状的问题(How)的解决方法和上述方法大同小异。一般来说,对于这类问题,需要寻找的关系是动词和描述这个动作方式的副词或者副词词组。文献(Girju et al. 2003)采用机器学习的方法在文本中寻找满足描述方式关系的内容,提取的属性包括:在训练集合中的计数;在语法树中副词上层的结构,即该副词是修饰动词、名词或者形容词;作为副词的可能性;动词副词之间的距离;副词之前的词的词性;副词之后的词的词性;该词是否以“ly”结尾等。通过一些训练数据,学习这些属性的分布,获得一个朴素贝叶斯分类器,从而判断文本中所有这类关系,帮助回答关于方式的问题。

上述解决特殊类问题的方法类似,它们首先把这类问题对应到一种关系,然后通过机器学习的方法从文档集中找出满足这类关系的文本内容,从而和问题相匹配。

### 三、通过系综方法构建问答系统

俗话说:“三个臭皮匠,顶个诸葛亮”。系综方法就是把多种方法综合使用,从而获得比任何单一方法更好的效果。近年来,系综学习方法已经成为机器学习领域最热门的研究方向之一。在问答系统的设计中,也有采用系综学习方法的应用。

在问答系统中有多个层次采用系综方法。最高的层次类似于一个元搜索引擎,把各种不同问答系统的结果做归并。和元搜索引擎一样,由于获得信息量有限,这种归并是不完全的。更好的一种方法是在多个层次上做归并。文献(Chu C et al. 2003)就是使用多种资源和多种策略归并,以提高回答的效果。该文采用 WordNet、Cyc、Web 等多种资源,采用基于知识的、基于统计的、定义类的和基于 KSP 的四种策略来完成问答系统的构建。归并的层次也不是单一的,而是多层次的。和上面描述的问题分析、段落检索和答案抽取三个模块相一致,这种方法也采取了三个层次上

的归并。

在问句分析层次,基于统计的方法和基于知识的方法各有一套问句类型的定义,通过两个子系统进行分类,如果发现两个系统分类获得的问句类型是相容的,就不需要做进一步处理。如果问句类型不能相容,就需要保留两种分类的结果。在段落检索层次,基于统计的方法和基于知识的方法都会检索获得一些文档,这里把基于知识方法策略认为最相关的前 10 篇文档加入基于统计获得的文档集中。在答案抽取层次,会把两个系统获得结果通过投票决定最后的答案。通过实验,多种策略的归并结果会比单个策略取得更好的结果。

以上介绍了在传统三大流程以外的其他方法,这些方法都能在不同方面和不同程度上提高问答系统的性能,以满足用户的需求。

## 第六节 问答系统的评测

作为一个系统,在工程学上往往要评价它的好坏,达到怎样的程度,这时候就需要一些评价指标。这些指标可以作为参照标准帮助系统设计者不断改善系统的性能。本节首先介绍 TREC 问答系统评测,然后给出系统评测的一些常用指标。

### 一、TREC 问答系统评测

为促进问答系统的研究,自 1999 年起 TREC 会议开始组织问答系统评测。TREC8 和 TREC2000 的评测问题只包括事实性的问题,它是从微软的百科全书 Encarta 和在线问答系统 START 中提取的。当时对于答案的要求不高,要求返回答案串的长度不超过 50 字节和 250 字节(分别是两个子任务),只要在答案串中包含了正确答案就可以。从 TREC2001 开始,把整个问答任务划分成三个子任务:主任务、列表类任务和上下文相关任务。其中主任务包含事实性的问题和定义性的问题,列表类任务要求给出一系列答案构成的列表,上下文相关任务是把问题集结成为一个个的主题之下。从 TREC2003 开始,把任务分成主任务和段落任务,其中段落任务沿袭以前的传统,要求返回一定长度限制(250 字节)以内的答案串,而主任务中又有三类问题,即事实类问题、列表类问题和定义类问题。对于前两类问题,要求回答的答案是准确的字符串,不得包含其他的成分。2004 年,一个明显的变化是所有的问题都是上下文相关的了,也就是集结于一个主题之下的。2005 年又增加了两个子任务,即段落检索子任务和关系类问题子任务,前者的提出是为了评估中间过程即段落检索模块的性能,后者的提出是为了鼓励对多个实体之间关系问题的研究。近年来在 TREC 问答类任务中表现比较好的研究机构包括语言计算公司(LCC),美国麻省理工大学(MIT)和新加坡国立大学(NUS)。

从 TREC 问答任务的发展历程可以看出,评测和系统性能是一个互相促进的关

系,最初的评测只要求能够包含答案的文本段落就可以了,而且问题类型只是最简单的事实类的问题,随着参加评测系统性能的提高,新的问题类型如定义类问题、列举类问题、关系类问题、上下文相关类问题都加进来了,答案也由一个文本段落变成完全匹配的字串,问题的难度和答案的要求也提高了。

## 二、问答系统评测指标

在 TREC 的问答任务中,主任务的三类问题,即事实类问题,列表类问题和定义类问题,分别有不同的评判方法和指标。

事实类问题要求回答给出答案和它的出处(该答案从哪篇文档获得)。答案正确当且仅当答案的字符串和标准答案字符串完全匹配并且出处是属于合法的出处集合的。如果答案正确但是出处不正确,称这个答案为“不被支持的”;如果答案是包含了标准答案串的一个串或者只包含标准答案串的一部分,那么认为这个答案“不精确匹配”;其他的情况就直接称为“不正确”。另一种情况是某个问题本身在文档集中找不出正确答案,此时应该返回 NIL。如果系统返回的是 NIL,则认为结果正确,如果系统还返回一个答案和相关出处,则认为结果“不正确”。回答事实类问题的总体性能由准确率(Accuracy)进行判断,即在所有问题中能够正确回答的比例。在此之前,TREC 是允许提供多个答案的,如提供 5 个答案,评测的方法就是计算这 5 个中第一个正确的倒数平均值。对这两个指标形式化表示为

$$p = \frac{|\{q \mid q \in Q \wedge \text{Answer}(q) = \text{correct}\}|}{N}$$

$$\text{MRR} = \frac{\sum_{i=1}^N \frac{1}{\text{Rank}(Q_i)}}{N}$$

其中, $p$  表示回答的准确率, $N$  表示总的问题的数目, $\text{Rank}(Q_i)$  表示第  $i$  个问题的第一个正确的序号,如果系统没有给出正确的答案,那么该值为正无穷大。

列表类问题要求返回的格式和事实类问题是一样的,不同的是它需要返回多个这样的答案,因此需要把系统返回的结果集合和标准答案集合做一个比较,这和传统信息检索里面返回结果的评测非常相似,不同在于信息检索返回的是文档集合,即答案集合。类似的,通过查准率和查全率可以判断对于列表类问题回答的好坏程度,为了统一表示查准率和查全率,还可用这两者的调和平均值  $F$  表示。

$$\text{precision} = \frac{|\text{Ra}|}{|R|}$$

$$\text{recall} = \frac{|\text{Ra}|}{|A|}$$

$$F = \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

其中, $\text{Ra}$  表示返回结果中正确的部分, $R$  表示系统返回的所有结果集, $A$  表示所有正

确的结果。

定义类问题的评测十分复杂,与事实类和列表类的问题不同的是:它返回的结果是由很多文本的片断组合而成的。因此,不能用简单的字符匹配的方法对它进行评测,当前对于这类问题的评测主要还是采用人工的方法。TREC 对这类问题评测的基本流程是:首先由各个参赛队提交结果,由工作人员从所有的结果中提出能够回答问题的一些“得分点”,并把得分点分为两种,一种是包含了关键信息的“关键得分点”(Vital Nugget),另一些是正确但是并不非常重要的“非关键得分点”(Okay Nugget)。对系统的评分主要由命中的关键得分点的数目和答案的长度综合而成。和信息检索评测类似的,也计算查准率和查全率,不过这两个概念和原始的概念有一些出入,定义为

$$\begin{aligned} \text{recall} &= \frac{|\text{match\_vital\_nuggets}|}{|\text{vital\_nuggets}|} \\ \text{precision} &= \begin{cases} 1 & \text{length} < \text{okay\_length} \\ 1 - \frac{\text{length} - \text{okay\_length}}{\text{length}} & \text{otherwise} \end{cases} \\ \text{okay\_length} &= C \times |\text{total\_nuggets}| \\ F &= \frac{(\beta^2 + 1) \times \text{precision} \times \text{recall}}{\beta^2 \times \text{precision} + \text{recall}} \end{aligned}$$

其中,match\_vital\_nuggets 表示在回答中能找到的“关键得分点”集合,vital\_nuggets 表示总共“关键得分点”的集合,二者的商就是系统可以找到的关键得分点占总的比例。length 表示返回答案的长度,C 是一个常数,这里取 100,表示一个得分点允许回答的字节数,total\_nuggets 表示总“得分点”的集合。okay\_length 是对于一个问题的合理的回答长度,给每个“得分点”100 字节的长度限制。如果回答不超过这个长度限制,那么认为准确率是 1,单纯从覆盖率来测试性能,如果超过这个长度限制,那么就会扣分。F 就是查准率和查全率的加权的调和平均值, $\beta$  取值为 5,即认为 recall 更重要。

在 TREC 的评测中,对定义类问题的评测并不是自动的,需要大量的手工参与,这不仅需要耗费大量的人力,也有可能带入一些主观的因素。对于这类问题的评测,有研究者试图通过自动的方法完成。例如,文献(Lin and Demner-Fushman 2005)提出了一种简单的评测方法,它需要事先构建一个“关键得分点”的集合,对于每个得分点,和系统返回的答案相比较,计算连续最大长度匹配。对于匹配的词,如一个得分点是“ABC D”,系统返回是“BEBC D F”,那么最长长度匹配的串就是“BC D”。对于匹配串中的每一个词,计算一个分值,分值计算方式可以是计算匹配串内每一个词的 IDF 的叠加,因为 IDF 越大说明这个词汇越不可能出现,如果匹配说明系统的效果越好。最后叠加所有“得分点”的得分获得总得分。通过把该方法和手工评测方法比较,比较各个参赛队排名后的序列相似度(Kendall's T)验证了这种方法的有效性。在实际评测的过程中发现,不同的评审员对同一个答案段落的“关键得分点”和

“非关键得分点”的意见会有出入,为了解决这个问题,文献(Lin et al. 2006)借用自动摘要领域的一种办法,对每个得分点给予一个权重,这个权重是它被认为是“关键得分点”的次数和被认为“关键得分点”次数最多的“得分点”的次数的商,因此,任何一个“得分点”的权重都为 0~1。

## 第七节 实例:天网开放域问答系统

在 TREC2005 年的问答评测中,北京大学网络与分布式实验室设计和实现了天网开放域问答系统并参加了评测。通过实践,我们应用了问答系统研究中的一些技术,并进行了一些创新。本节将以这个系统为例,介绍如何应用前面几节介绍的技术和方法构建一个完整的问答系统。

为了提高系统的性能,利用了很多的外部资源,系统的体系结构如图 13-4 所示。与图 13-3 相比,该图显得复杂一些,这是因为在实际系统构建时需要考虑更多的细节问题。

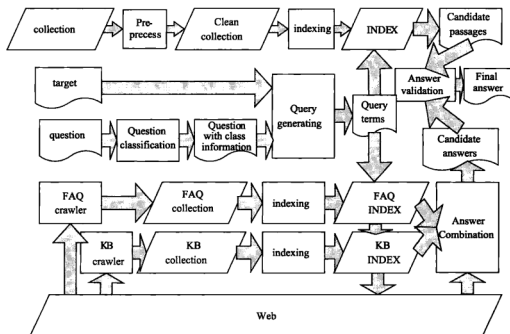


图 13-4 天网开放域系统的体系结构

对于 TREC 提供的数据,首先要进行数据预处理,这包括解析文档,去掉一些标签和停用词,对词进行去词缀操作(Stem)等。对预处理后的文档集,可以建立倒排索引。

另外,搜集了大量的常见问题解答作为 FAQ 库,这些常见问题解答有两个来

源:一是抓取在线的常见问题解答库(<http://www.faqs.org>),二是通过在搜索引擎中检索,如 FAQ 之类的词得到可能包括 FAQ 内容的网站和网页,然后从中运用正则表达式和一些其他的规则提取有用的问题-答案对作为基础资源。

还有一部分基础资源在系统中叫做“知识库”,为了构建这个知识库,利用网上的一些资源,如 CIA 的 factbook(<http://www.cia.gov/cia/publications/factbook>),美国 50 州网站(<http://www.50states.com>)和人物传记网站(<http://www.biography.com>)。这些资源的数据质量比较高,所包含的噪音小。利用一些手工编写的 pattern 提取网页中的有意义的数据,把这些数据组织成三元组〈实体,属性,值〉的形式。并将这些三元组建立索引,便于问句到来时进行查询。

上述工作可以称为离线操作,因为它们可以在问题到来之前完成。

当系统接受一个问句的时候,首先需要对问句进行分析。我们并没有特别做消解指代的工作,因为 TREC 的问题已经很明确的包含问句的目标和问句内容两个部分,大部分在问题中出现的代词都是对目标而言的。对问句进行分类,借用文献(Li et al. 2002)的分类体系结构,采用的是 SVM 的分类方法,这样每一个问句都会有一个类标签。

在文档检索模块,通过迭代获得布尔查询。需要注意的是,这里产生的布尔查询不仅用于查询文档集,也会拿来查询常见问题解答库,知识库和 Web。

检索获得了答案候选段落后,需要从中抽取候选的答案。采用比较简单的精细命名实体识别的方法,具体来说,对于实体进行问句类型和实体类型的匹配。此外,还通过 WordNet 和 Web 的资源获得一些实体的列表,协助匹配。对于某些类型的答案,如数量、日期等,采用预先定义的正则表达式进行匹配并提取。对于 Web 上获得的段落,也采取同样的方法,提取候选答案,在常见问题解答和知识库里面提取答案,赋予一定的权值加入候选答案集,然后对候选答案进行计数,计数比较大的答案,认为它们的置信程度比较高,作为正确答案。从而完成了答案的提取和验证。

表 13-2 显示了天网开放域系统在 TREC2005 中的表现,由于采用的技术比较简单,所以效果不理想。

表 13-2 天网开放域系统在 TREC2005 中的表现

	Best	Median	Worst	Ours
factoid question	0.713	0.152	0.014	0.108
list question	0.468	0.053	0	0.053
other question	0.248	0.156	0	0.025
per series	0.534	0.123	0.008	0.075



## 第八节 小 结

本章主要介绍了开放域问答系统的发展历史,应用背景,实际系统构建时所用的主要技术与方法等。

开放域问答系统在计算机诞生之初就已经被图灵提出来。随着网上信息查找和信息获取需求的增大,以及信息检索和信息提取技术的发展,问答系统的研究得到了快速的发展。TREC 设置的问答任务,已经成为 TREC 保留时间最长的任务之一。

目前,大多数开放域问答系统都包括问题分析、文档段落的检索、答案的抽取和验证三个顺序的过程。问题分析的主要任务包括指代消解,问句分类。文档段落检索就是要从问句到查询,并且根据查询从文档集中得到可能包含正确答案的文档集或者段落集。答案的抽取和验证就从候选的文档集或者段落集中提取正确的答案。

问答系统性能的改进方法,包括利用外部资源(如 WordNet、Web 数据、常见问题解答库等)研究特定类型问题的特定解决方法,以及采用系综方法构造问答系统等。

问答系统评测是一项很重要的工作,TREC 为此做出了很大的贡献,它不断改变评测的子任务和评测指标,使之逐渐接近现实系统的要求。本章最后介绍了一个我们自己设计和实现的问答系统。

本章内容取材于何靖等的问答系统综述报告(何靖等 2010)。

## 参考文献

- 卜东波. 2000. 聚类/分类理论研究及其在文本挖掘中的应用. 中国科学院博士学位论文.
- 车万翔, 刘挺等. 2004. 基于改进编辑距离的中文相似句子检索. 高技术通讯, 14(7): 34-37.
- 陈鼎. 2008. 网络资源的名字特征及其在资源组织中的应用研究. 北京大学博士学位论文.
- 陈鼎, 彭波等. 2005. 一种词汇共现算法及其共现词对检索系统排序的影响. 清华大学学报(自然科学版), 45(S1): 1857-1860.
- 陈菊红. 2009. 搜索引擎返回结果聚类技术的研究与实现. 西南交通大学硕士学位论文.
- 陈燕, 陶丹等. 2002. 传播学研究方法. 北京: 科学出版社.
- 邓志鸿, 唐世渭等. 2002. Ontology 研究综述. 北京大学学报(自然科学版), 38(5): 730-738.
- 冯是聪. 2003. 中文网页自动分类技术研究及其在搜索引擎中的应用. 北京大学博士学位论文.
- 冯是聪, 张志刚等. 2004. 一种中文网页自动分类方法的实现及其应用. 计算机工程, 30(5): 19-20.
- 郭化楠, 李静静等. 2004. 如何制作 CWT100g 的 topics. 北京大学计算机系网络与分布式系统实验室.
- 何靖. 2006. 一种问答式检索系统布尔查询生成方法. 全国搜索引擎和网上信息挖掘学术研讨会.
- 何靖, 陈鼎, 闫宏飞. 2010. 开放域问答系统研究综述. 第六届全国信息检索会议(CCIR2010).
- 黄菁堂, 吴立德. 1998. 基于向量空间模型的文档分类系统. 模式识别与人工智能, 11(2): 147-153.
- 黄连恩. 2008. 历史网页的持续收藏及其再访问的关键技术研究. 北京大学博士学位论文.
- 江玉婷, 陈光华. 1998. TREC 现况及其对信息检索研究之影响. 图书与信息学刊, 29(4): 36-59.
- 雷鸣. 2000. 一个大规模、高性能的搜索引擎系统及索引和检索子系统的实现. 北京大学硕士学位论文.
- 刘开瑛. 2000. 中文文本自动分词和标注. 北京: 商务印书馆.
- 刘群, 李素建. 2002. 基于《知网》的词汇语义相似度计算. 第三届汉语词汇语义学研讨会.
- 刘远超, 王晓龙等. 2006. 文档聚类综述. 中文信息学报, 20(3): 55-62.
- 马亮, 陈群秀. 2002. 智能 Web 中文主题信息收集系统 IRobot 的设计. 中文信息学报, (16): 23-29.
- 彭波. 2004a. 搜索引擎检索系统的效率优化和效果评估研究. 北京大学博士学位论文.
- 彭波. 2004b. 搜索引擎中倒排文件索引的缓存机制. Pku\_Cs\_Net\_Tr2004005. <http://162.105.80.88/crazys-ite/home/report/upload/1820180625.doc>.
- 单松巍. 2003. 搜索引擎的日志分析: 方法、技术和应用. 北京大学硕士学位论文.
- 宋炜, 张铭. 2004. 语义网简明教程. 北京: 高等教育出版社.
- 天网. 2012. 北京大学天网中英文搜索引擎. <http://e.pku.edu.cn>.
- 奚婷. 2008. 搜索引擎结果的聚类系统研究. 西安交通大学硕士学位论文.
- 谢正茂. 2003. Web 数据模型以及获取、存储方法研究. 北京大学硕士学位论文.
- 闫宏飞. 2002. 可扩展 Web 信息搜集系统的设计、实现与应用初探. 北京大学博士学位论文.
- 管红英. 2002. 基于实体属性的中文网页检索研究. 北京大学博士学位论文.
- 张志刚. 2004. 基于网页的信息系统的一种预处理过程. 北京大学硕士学位论文.
- 张志刚, 陈静等. 2004. 一种 HTML 网页净化方法. 情报学报, 23(4): 387-393.
- 赵江华. 2002. 天网高性能分布式检索系统的设计与实现. 北京大学硕士学位论文.
- 中国互联网络信息中心(CNNIC). 2012. 第 29 次中国互联网络发展状况调查报告. <http://www.cnnic.net.cn>.
- 周登鹏. 2007. 搜索引擎搜索结果的聚类研究. 上海交通大学硕士学位论文.
- ADL. 2004. Alexandria Digital Library Project. <http://alexandria.sdc.ucsb.edu>.

- Agichtein E, Lawrence S, et al. 2004. Learning to find answers to questions on the Web. *ACM Trans. Internet Technol.*, 4(2):129-162.
- Almeida V et al. 1996. Characterizing reference locality in the WWW. *Proceedings of the 4th International Conference on Parallel and Distributed Information Systems*. Miami Beach, FL, USA.
- Altingovde I S, Özcan R, et al. 2009. Exploiting query views for static index pruning in web search engines. *Proceedings of the 18th ACM Conference on Information and Knowledge Management*. ACM.
- Anh V N, Moffat A. 2002. Impact transformation: effective and efficient web retrieval. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Tampere, Finland, ACM Press.
- Anh V N, Moffat A. 2004. Index compression using fixed binary codewords. *Proceedings of the 15th Australasian Database Conference-Volume 27*. Australian Computer Society, Inc.
- Anh V N, Moffat A. 2005. Inverted index compression using word-aligned binary codes. *Information Retrieval*, 8(1):151-166.
- Anh V N, Moffat A. 2006. Improved word-aligned binary compression for text indexing. *IEEE Transactions on Knowledge and Data Engineering*, 18(6):857-861.
- Anh V N, Moffat A. 2006. Pruned query evaluation using pre-computed impacts. *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.
- Anh V N, Moffat A. 2006. Pruning strategies for mixed-mode querying. *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*. Arlington, Virginia, USA.
- Anh V N, Moffat A. 2006. Structured index organizations for high-throughput text querying. *String Processing and Information Retrieval*, Springer.
- Anh V N, Moffat A. 2010. Index compression using 64 bit words. *Softw. Pract Exper*, 40(2):131-147.
- Anh V N, De Kretser O, et al. 2001. Vector-space ranking with effective early termination. *Proceedings of the 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.
- Archive I. 2012a. Internet Archive. <http://www.archive.org>.
- Archive I. 2012b. Internet Archive. <http://www.archive.org/about/faqs.php>.
- Ashish N, Knoblock C. 1997. Wrapper generation for semistructured Internet sources. *Proc. PODS/SIGMOD'97*.
- Attar R A, Fraenkel S. 1977. Local Feedback in Full-Text Retrieval Systems. *J. ACM*, 24(3):397-417.
- Baeza-Yates R, Ribeiro-Neto B. 1999. *Modern Information Retrieval*. London: Addison-Wesley-Longman.
- Bahle D, Williams H E, Zobel J. 2002. Efficient phrase querying with an auxiliary index. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.
- Baldi P, Frasconi P, Smyth P. 2003. *Modeling the Internet and the Web, probabilistic methods and algorithms*. Hoboken: John Wiley & Sons.
- Beefenman D, Berger A, et al. 1997. A model of lexical attraction and repulsion. *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics*.
- Bentley J L, Sleator D D, et al. 1986. A locally adaptive data compression scheme. *Communications of the ACM*, 29(4):320-330.
- Bergman M K. 2000. The deep web: surfacing hidden value. *The Journal of Electronic Publishing*, 7(1).
- Berry M W, Dumais S T, et al. 1995. Using linear algebra for intelligent information retrieval. *SIAM Review*, 37(4):573-595.
- Bilotti M W, Ogilvie P, et al. 2007. Structured retrieval for question answering. *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Amsterdam.

- The Netherlands. ACM.
- Blanco R, Barreiro A. 2005. A Software Architecture for Effective Document Identifier Reassignment. Springer Berlin / Heidelberg. 3643;254-262.
- Blanco R, Barreiro A. 2005. Characterization of a simple case of the reassignment of document identifiers as a pattern sequencing problem. Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM.
- Blanco R, Barreiro A. 2005. Document Identifier Reassignment through Dimensionality Reduction. Springer Berlin / Heidelberg. 3408;375-387.
- Blanco R, Barreiro A. 2006. TSP and cluster-based solutions to the reassignment of document identifiers. Information Retrieval, 9(4);499-517.
- Blanco R, Barreiro A. 2007a. Boosting static pruning of inverted files. Proceedings of SIGIR. ACM.
- Blanco R, Barreiro A. 2007b. Static pruning of terms in inverted files. Advances in Information Retrieval; 64-75.
- Blanco R, Barreiro A. 2010. Probabilistic static pruning of inverted files. ACM Transactions on Information Systems, 28(1);1-33.
- Blandford D, Blelloch G. 2002. Index compression through document reordering. Data Compression Conference Proceedings. DCC 2002, IEEE.
- Bookstein A, Klein S T, et al. 1994. Markov models for clusters in concordance compression. Data Compression Conference. Proceedings. DCC'94. IEEE.
- Bookstein A, Klein S T, et al. 1997. Modeling word occurrences for the compression of concordances. ACM Transactions on Information Systems (TOIS), 15(3);254-290.
- Brin S, Page L. 1998. The anatomy of a large-scale hypertextual Web search engine. Computer Networks and ISDN Systems, 30(1-7);107-117.
- Broder A Z, Carmel D, et al. 2003. Efficient query evaluation using a two-level retrieval process. Proceedings of the 12th International Conference on Information and Knowledge Management. ACM.
- Broder A, Kumar R, et al. 2000. Graph structure in the web; experiments and models. Proceedings of the 9th World-Wide Web Conference. Amsterdam.
- Bruce Ct, Donald M, et al. 2009. Search Engines; Information Retrieval in Practice. London; Pearson Education.
- Buckley C, Lewit A F. 1985. Optimization of inverted vector searches. Proceedings of the 8th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM.
- Buckley C, Voorhees E M. 2000. Evaluating evaluation measure stability. Proceedings of the 23rd International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2000), Athens, Greece.
- Buckley C, Voorhees E M. 2004. Retrieval evaluation with incomplete information. Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Sheffield, United Kingdom. ACM.
- Burrows M, Wheeler D J. 1994. A block-sorting lossless data compression algorithm. System Research Center, Palo Alto, USA.
- Büttcher S C, Clarke L A. 2007. Index compression is good, especially for random access. Proceedings of the 6th ACM Conference on Conference on Information and Knowledge Management. ACM.
- Büttcher S C, Clarke L A, et al. 2006. The TREC 2006 terabyte track. Proceedings of TREC. Citeseer.
- Campinas S, Delbru R, et al. 2011. SkipBlock; self-indexing for block-based inverted list. Advances in Information Retrieval; 555-561.
- Carmel D, Cohen D, et al. 2001. Static index pruning for information retrieval systems. Proceedings of the 24th

- Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM, Carrot2. 2012. Carrot2 系统. <http://Project.carrotz.org>.
- Chakrabarti K, Chaudhuri S, et al. 2011. Interval-based pruning for top-k processing over compressed lists. 2011 IEEE 27th International Conference on Data Engineering (ICDE). IEEE.
- Chakrabarti S, Joshi M, et al. 2001. Enhanced topic distillation using text, markup tags, and hyperlinks. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New Orleans, LA.
- Chang K C, He C B, et al. 2005. Toward Large Scale Integration; Building a MetaQuerier over Databases on the Web. Proceedings of the Second Conference on Innovative Data Systems Research (CIDR 2005). Asilomar, Ca.
- Charles L, Clarke Q, Buettcher S, et al. 2010. Information Retrieval; Implementing and Evaluating Search Engines. Cambridge; The MIT Press.
- Cheng C S, Chung C P, et al. 2006. Fast query evaluation through document identifier assignment for inverted file-based information retrieval systems. Information Processing & Management, 42(3):729-750.
- Cheng C S, Shann J J J, et al. 2006. Unique-order interpolative coding for fast querying and space-efficient indexing in information retrieval systems. Information Processing & Management, 42(2):407-428.
- Chidlovskii B, Rencancio C, et al. 1999. Semantic cache mechanism for heterogeneous Web querying. Computer Networks, 31(11):1347-1360.
- Cho J. 2002. Crawling the Web; Discovery and maintenance of large-scale Web data. Stanford University. PhD.
- Cho J, Molina H G. 2000. The evolution of the Web and implications for an incremental crawler. Proceedings of 26th International Conference on Very Large Databases (VLDB) Cairo, Egypt.
- Christopher D, Manning P, et al. 2008. Introduction to Information Retrieval. England; Cambridge University Press.
- ChSeg. 2003. Chinese Segmentation. <http://net.pku.edu.cn/~webg/src/ChSeg>.
- Chu C J, Prager J, et al. 2006. Semantic search via XML fragments; a high-precision approach to IR. Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle. Washington, USA, ACM.
- Chu C, Czuba K L, et al. 2003. In question answering, two heads are better than one. Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology. Edmonton, Canada.
- Church K, Hanks W P. 1990. Word association norms, mutual information, and lexicography. Comput Linguist, 16(1):22-29.
- Cleverdon C. 1997. The Cranfield tests on index language devices. Readings in Information Retrieval. Morgan Kaufmann Publishers Inc., San Francisco, CA.
- Collins T K, Callan J, et al. 2004. The effect of document retrieval quality on factoid question answering performance. Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Sheffield, United Kingdom. ACM.
- Comer D. 1979. Ubiquitous B-tree. ACM Computing Surveys (CSUR), 11(2):121-137.
- Cormack G V, Palmer C R, et al. 1998. Efficient construction of large test collections. Proceedings of the 1998 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98). Melbourne, Vic. Aust, ACM, New York, NY, USA.
- Cormen T H, Leiserson C E, et al. 2001. Introduction to Algorithms. 2nd ed. Cambridge; The MIT Press.
- Craswell N, Hawking D, et al. 2003. Overview of the TREC-2003 Web Track. Proceedings of TREC 2003.

- Gaithersburg, Maryland USA.
- Cui H, Kan M Y, et al. 2004a. Unsupervised learning of soft patterns for generating definitions from online news. Proceedings of the 13th international conference on World Wide Web. New York, NY, USA, ACM.
- Cui H, Kan M Y, et al. 2004c. A comparative study on sentence retrieval for definitional Question Answering. SIGIR Workshop on Information Retrieval for Question Answering. Sheffield, UK.
- Cui H, Li K, et al. 2004b. National University of Singapore at the TREC-13 Question Answering. Proceeding of the 13th Text Retrieval Conference. Gaithersburg; NIST Special Publication.
- Cui H, Sun R, et al. 2005. Question answering passage retrieval using dependency relations. Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Salvador, Brazil. ACM.
- Cummins R, O'Riordan C, et al. 2010. An analysis of learned proximity functions. Proceedings of Personalization and Fusion of Heterogeneous Information.
- DCMI. 2008a. Dublin Core Metadata Element Set, Version 1. 1; Reference Description. <http://dublincore.org/documents/dces>.
- DCMI. 2008b. DCMI type vocabulary. <http://dublincore.org/documents/dcmi-type-vocabulary>.
- DDC. 2008. Dewey Decimal Classification. <http://www.oclc.org/dewey>.
- Dean J. 2009. Challenges in building large-scale information retrieval systems; invited talk. Proceedings of the Second ACM International Conference on Web Search and Data Mining. ACM.
- Dean J A, Haahr P G, et al. 2004. Multi-stage query processing system and method for use with tokenspace repository. Google Patents.
- Deutsch L P. 1996. DEFLATE compressed data format specification version 1. 3. <http://gama. tools. ietf. org/html/rfc1951>.
- Deutsch L P. 1996. GZIP file format specification version 4. 3. <http://merlot. tools. ietf. org/html/rfc1952>.
- Ding S, Suel T. 2011. Faster top-k document retrieval using block-max indexes. Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information. ACM.
- Ding S, Attenberg J, et al. 2010. Scalable techniques for document identifier assignment in inverted indexes. Proceedings of the 19th International Conference on World Wide Web. ACM.
- Dublincore. 2003. Dublin Core Metadata Element Set, Version 1. 1; Reference Description. <http://dublincore.org/documents/dces>.
- EAD. 2002. Encoded Archival Description (EAD). Official EAD Version 2002 Web Site. <http://www.loc.gov/ead>.
- Echihabi A, Marcu D. 2003. A noisy-channel approach to question answering. Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. Sapporo, Japan.
- Elias P. 1975. Universal codeword sets and representations of the integers. IEEE Transactions on Information Theory, 21(2):194-203.
- eTesting L. 2000. Google Web Search Engine Evaluation. <http://www.veritest.com/clients/reports/google/google.pdf>.
- Fagin R. 2003. Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences, 66(4):614-656.
- Fang H C, Zhai. 2006. Semantic term matching in axiomatic approaches to information retrieval. Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, Washington, USA.

- Fay C, Dean J, Ghemawat S. 2006. Bigtable: A distributed storage system for structured data. 7th Symposium on Operating Systems Design and Implementation(OSDI), Seattle, WA.
- Fleischman M, Hovy E, et al. 2003. Offline strategies for online question answering: answering questions before they are asked. Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1. Sapporo, Japan.
- Fraenkel A S, Klein S T. 1985. Novel compression of sparse bit-strings-Preliminary report. Combinatorial Algorithms on Words, 12(4):169-183.
- Garcia M H. 2008. Web information management: past, present and future. Proceedings of the international Conference on Web Search and Web Data Mining. ACM.
- Garcia S. 2007. Search Engine Optimization Using Past Queries. School of Computer Science and IT, PhD.
- Ghemawat S H, Gobiuff, et al. 2003. The Google file system. ACM SIGOPS Operating Systems Review. ACM.
- Girju R. 2003. Automatic detection of causal relations for Question Answering. Proceedings of the ACL 2003 Workshop on Multilingual Summarization and Question Answering-Volume 12. Sapporo, Japan.
- Girju R, Putcha M, et al. 2003. Discovery of manner relations and their applicability to Question Answering. Proceedings of the ACL 2003 Workshop on Multilingual Summarization and Question Answering-Volume 12. Sapporo, Japan.
- Golomb S. 1966. Run-length encodings. IEEE Transactions on Information Theory, 12(3):399-401.
- Google. 2012. Google Search Engine. <http://www.google.com>.
- Gruhl D, Chavet L, et al. 2004. How to build a WebFountain: An architecture for very large-scale text analytics. IBM Systems Journal, 43(1):64-77.
- Hammer J, et al. 1997. Extracting Semistructured Information from the Web. Workshop on Management of Semistructured Data. Tucson, Arizona.
- Han J, Micheline K. 2007. 数据挖掘概念与技术. 2版. 北京:机械工业出版社.
- Hatcher E, Gospodnetic O. 2005. Lucene in Action. New York: Manning Publications
- Hawking D, Craswell N, et al. 1999. Results and challenges in Web search evaluation. Computer Networks, 31(11):1321-1330.
- Hawking D, Craswell N, et al. 2001. Measuring search engine quality. Information Retrieval, 4(1):33-59.
- He J, Chen C, et al. 2005. Peking University at the TREC-2005 Question and Answering Track. 14th Text Retrieval Conference.
- Heaps H S. 1978. Information Retrieval: Computational and Theoretical Aspects. Orlando: Academic Press, Inc.
- Heinz S, Zobel J. 2003. Efficient single-pass index construction for text databases. Journal of the American Society for Information Science and Technology, 54(8):713-729.
- Héman S. 2005. Super-Scalar Database Compression between RAM and CPU Cache. University of Amsterdam. Master's Thesis.
- Hendrix G G, Sacerdoti E D, et al. 1978. Developing a natural language interface to complex data. ACM Trans. Database Syst, 3(2):105-147.
- HTMLTidy. 2004. HTML Tidy Library Project. <http://tidy.sourceforge.net>.
- Hull D. 1993. Using statistical testing in the evaluation of retrieval experiments. Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA.
- Ibiblio. 2008. Ibiblio Public's Library and Digital Archive. <http://www.ibiblio.org/collection>.

- Ilyas I F, Beskales G, et al. 2008. A survey of top-k query processing techniques in relational database systems. *ACM Comput. Surv.*, 40(4):111-118.
- IMDB. 2008. Internet Movie Database. <http://www.imdb.com>.
- InfoMall. 2012. China Web InfoMall(中国互联网信息博物馆). <http://www.infomall.cn>.
- Jarvelin K, Kekalainen J. 2000. IR evaluation methods for retrieving highly relevant documents. *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Athens, Greece. ACM.
- Jeon J, Croft W B, et al. 2005. Finding similar questions in large question and answer archives. *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*. Bremen, Germany. ACM.
- Jian R. 1991. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation, and Modeling*. New York: John Wiley & Sons.
- Jijkoun V, Rijke M D. 2005. Retrieving answers from frequently asked questions pages on the web. *Proceedings of the 14th ACM International Conference on Information and Knowledge Management*. Bremen, Germany. ACM.
- Jin S, Bestavros A. 2000. Sources and characteristics of Web temporal locality. *Proceedings of the 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication System*. San Francisco, CA, USA.
- Jonassen S, Bratsberg S. 2011. Efficient Compressed Inverted Index Skipping for Disjunctive Text-Queries. *Advances in Information Retrieval*:530-542.
- Jones P. 2001. Open(source)ing the Doors for Contributor-run Digital Libraries. *Communication of the ACM*, 44(5):45-46.
- Jonsson B T, Franklin M J, et al. 1998. Interaction of query evaluation and buffer management for information retrieval. *Proceedings of the ACM SIGMOD International Conference on Management of Data*. Seattle, WA, USA.
- Josuttis N M. 1999. *The C++ Standard Library A Tutorial Reference*. New York: Addison Wesley Longman, Inc.
- Kahle B. 1997. Preserving the Internet. *Scientific American*, 276(3):82-83.
- Kang I H, Kim G. 2003. Query Type Classification for Web Document Retrieval. *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Toronto, Ont, Canada.
- Katz B, Lin J. 2003. Selectively using relations to improve precision in question answering. *EACL 2003 Workshop on Natural Language Processing for Question Answering*. Budapest, Hungary.
- Kleinberg J M. 1998. Authoritative sources in a hyperlinked environment. *Proceedings of the 9th Annual ACM SIAM Symposium on Discrete Algorithms*. San Francisco, CA, USA.
- Ko J, Nyberg E, et al. 2007. A probabilistic graphical model for joint answer ranking in question answering. *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Amsterdam, The Netherlands. ACM.
- Lackie R J. 2006. Those Dark Hiding Places; The Invisible Web Revealed. <http://library.rider.edu/scholarly/rlackie/Invisible/Inv-Web-Main.html>.
- Lam H T, Perego R, et al. 2010. On Using Query Logs for Static Index Pruning. *2010 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. IEEE.
- Lavee G, Lempel R, et al. 2011. Inverted index compression via online document routing. *Proceedings of the*



- 20th International Conference on World Wide Web. ACM.
- Law C. 2001. PANDORA The Australian Electronic Heritage in a Box. International Preservation News; 13-17.
- LCC. 2008. Library of Congress Classification Outline. <http://www.loc.gov/catdir/cpsd/lcco>.
- Lei M, Wang J Y, et al. 2001. Improved relevance ranking in WebGather. Journal of Computer Science and Technology, 16(5):410-417.
- Lesk M. 1969. Word-word associations in document retrieval systems. American Documentation, 20(2):27-38.
- Lester N, Moffat A, et al. 2005. Space-limited ranked query evaluation using adaptive pruning. Web Information Systems Engineering-WISE 2005;470-477.
- Li X, Roth D. 2002. Learning question classifiers. Proceedings of the 19th International Conference on Computational Linguistics-Volume 1. Taipei, Taiwan, Association for Computational Linguistics.
- Li X, Shi Z. 2002. Innovating web page classification through reducing noise. Journal of Computer Science and Technology, 17(1):9-17.
- Light M, Mann G S, et al. 2001. Analyses for elucidating current question answering technology. Nat. Lang. Eng. , 7(4):325-342.
- Lin J, Katz B. 2003. Question answering from the web using knowledge annotation and knowledge mining techniques. Proceedings of the 12th International Conference on Information and Knowledge Management, New Orleans, LA, USA, ACM.
- Lin J D, Demner F. 2005. Automatically evaluating answers to definition questions. Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing. Vancouver, British Columbia, Canada, Association for Computational Linguistics.
- Lin J D, Demner F. 2006. Will pyramids built of nuggets topple over? Proceedings of the main conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics. New York; Association for Computational Linguistics.
- Lin S H, Ho J M. 2002. Discovering informative content blocks from Web documents. KDD-2002 Proceedings of the Eight ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Edmonton, Alberta, Canada; Association for Computing Machinery.
- Liu J, Lei M, et al. 2000. Digging for gold on the web; Experience with the WebGather. Proceedings of the 4th International Conference on High Performance Computing. The Asia-Pacific Region. IEEE Computer Society Press.
- Long X, Suel T. 2003. Optimized query execution in large search engines with global page ordering. Proceedings of the 29th International Conference on Very Large Data Bases. Volume 29, VLDB Endowment.
- Lu Z. 1999. Scalable distributed architectures for information retrieval. University of Massachusetts Amherst. PhD.
- Manber U. 1994. Finding similar files in a large file system. Usenix Winter 1994 Technical Conference. San Francisco.
- Mann G S. 2002. Fine-grained proper noun ontologies for question answering. COLING-02 on SEMANET; Building and Using Semantic Networks. Volume 11, Association for Computational Linguistics; 1-7.
- Markatos E P. 2001. On caching search engine query results. Computer Communications, 24(2):137-143.
- Moffat A, Zobel J. 1992. Parameterised compression for sparse bitmaps. Proceedings of the 15th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New York; ACM Press.
- Moffat A, Zobel J. 1996. Self-indexing inverted files for fast text retrieval. ACM Transactions on Information Systems, 14(4):349-379.

- Moffat A, Stuiver L. 1996. Exploiting clustering in inverted file compression. Data Compression Conference, 1996(DCC'96). Proceedings. IEEE.
- Moffat A, Stuiver L. 2000. Binary Interpolative Coding for Effective Index Compression. Information Retrieval, 3(1):25-47.
- Moffat A, Anh V N. 2006. Binary codes for locally homogeneous sequences. Inf. Process. Lett., 99(5): 175-180.
- Moldovan D, Pasca M, et al. 2003. Performance issues and error analysis in an open-domain question answering system. ACM Trans. Inf. Syst., 21(2):133-154.
- Moldovan D, Harabagiu S, et al. 2002. LCC tools for question answering. Proceedings of the 11th Text Retrieval Conference (TREC-11), Gaithersburg, MD.
- Moura E S d, Santos C F D, et al. 2008. Locality-Based pruning methods for web search. ACM Transactions on Information Systems, 26(2):1-28.
- Najork M, Heydon A. 2001. High-Performance Web Crawling. Compaq Systems Research Center.
- Najork M, Wiener J L. 2001. Breadth-first crawling yields high-quality pages. Proceedings of the 10th International Conference on World Wide Web. New York: ACM Press.
- Navarro G, De Moura E S, et al. 2000. Adding compression to block addressing inverted indexes. Information Retrieval, 3(1):49-77.
- Newman H. 2001. Application Performance and I/O. Storage, SW Expert. <http://swexpert.com/CB/SE.C11.APR.01.pdf>.
- Nguyen L T. 2009. Static Index Pruning for Information Retrieval Systems: A Posting-Based Approach. Proceedings of LIDS-IR. CEUR Workshop.
- Nie J Y, Gao J, et al. 2000. On the use of words and n-grams for Chinese information retrieval. Proceedings of the 5th International Workshop on Information Retrieval with Asian Languages. Hong Kong: ACM.
- Ntoulas A J, Cho. 2007. Pruning policies for two-tiered inverted index with correctness guarantee. Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM.
- OAI. 2003. Open Archives Initiative. <http://oai.granger.uiuc.edu>.
- Osiriski S, Stefanowski J, et al. 2004. Lingo: Search results clustering algorithm based on singular value decomposition. Intelligent information processing and web mining; proceedings of the International IIS/IIWPM'04 Conference. Zakopane, Poland.
- Page L, Brin S, et al. 1998. The PageRank Citation Ranking, Bringing Order to the Web. Stanford Digital Library Technologies Project.
- Pasca M, Harabagiu S M. 2001. The Informative Role of WordNet in Open-Domain Question Answering. The Proceeding of ECCAL 2001 Workshop on WordNet and Other Lexical Resources.
- Persin M, Zobel J, et al. 1996. Filtered document retrieval with frequency-sorted indexes. Journal of the American Society for Information Science, 47(10):749-764.
- Radev D R, Prager J, et al. 2000. Ranking suspected answers to natural language questions using predictive annotation. Proceedings of the 6th Conference on Applied Natural Language Processing. Seattle, Washington. Association for Computational Linguistics.
- Raghavan S, Garcia H M. 2001. Crawling the HiddenWeb. Proceedings of the 27th VLDB Conference. Roma, Italy.
- RFC2045, RFC2046, et al. 2004. The primary definition of MIME. <http://www.faqs.org/rfcs/index.html>.
- RFCs. 2004. RFCs Collection. <http://www.faqs.org/rfcs/index.html>.

- Rice R, Plaunt J. 1971. Adaptive variable-length coding for efficient compression of spacecraft television data. *Communication Technology. IEEE Transactions on*, 19(6):889-897.
- Rijsbergen C V. 1979. *Information Retrieval*. 2nd ed. London: Butterworths.
- Rijsbergen C V, Jones K S. 1975. Report on the need for and provision of an "ideal" information retrieval test collection. *British Library Research and Development Report 5266*, Computer Laboratory, University of Cambridge.
- Robertson S E, Walker S, et al. 1999. Okapi at TREC-7: automatic ad hoc, filtering, VLC and interactive track. *Nist Special Publication SP*, 21:253-264.
- Sadakane K, Imai H. 2001. Fast algorithms for k-word proximity search. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, E84-A(9):2311-2318.
- Salton G. 1971. *The SMART Retrieval System-Experiments in Automatic Document Processing*. Englewood Cliffs: Prentice Hall Inc.
- Salton G, Lesk M K. 1968. Computer evaluation of indexing and text processing. *J. ACM*, 15(1):8-36.
- Saracevic T. 1995. Evaluation of evaluation in information retrieval. *Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Seattle, WA, USA.
- Saraiva P C, et al. 2001. Rank-preserving two-level caching for scalable search engines. *24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. New Orleans, LA.
- Sata R. 2002. Presentation of the Internet Archive. *2nd ECDL Workshop on Web*.
- Schenkel R, Broschart A, et al. 2007. Efficient text proximity search. *String Processing and Information Retrieval*. New York: Springer.
- Scholer F, Williams H E, et al. 2002. Compression of inverted indexes For fast query evaluation. *Proceedings of the 25th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval-SIGIR'02*. New York: ACM.
- Searchenginewatch. 2004. Searchenginewatch. <http://www.searchenginewatch.com>.
- Sebastiani F. 1999. A tutorial on Automated Text Categorization. Analia Amandi and Alejandro Zunino (eds.), *Proceedings of ASAI-99, 1st Argentinian Symposium on Artificial Intelligence*. Buenos Aires, AR, 7-35.
- Sherman C, Price G. 2001. *The Invisible Web: Uncovering Information Sources Search Engines Can't See*. Medford: CyberAge Books.
- Shieh W. 2003. Inverted file compression through document identifier reassignment. *Information Processing & Management*, 39(1):117-131.
- Shivakumar N, Garca-Molina H. 1998. Finding near-replicas of documents on the web. *Proceedings of Workshop on Web Databases (WebDB'98)*.
- Silverstein C, Henzinger M, et al. 1998. Analysis of a Very Large AltaVista Query Log. *Hewlett Packard Laboratories SRC-TN-1998-014*, oct.
- Silvestri F. 2007. Sorting out the document identifier assignment problem. *Advances in Information Retrieval*, 101-112.
- Silvestri F, Perego R, et al. 2004. Assigning document identifiers to enhance compressibility of web search engines indexes. *Proceedings of the 2004 ACM Symposium on Applied Computing*. ACM.
- Silvestri F, Orlando S, et al. 2004. Assigning identifiers to documents to enhance the clustering property of full-text indexes. *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.
- Singhal A, Kaszkiel M. 2001. A case study in web search using TREC algorithms. *Proceedings of the 10th International Conference on World Wide Web*. Hong Kong: ACM.

- Skobeltsyn G,Junqueira F,et al. 2008. ResIn:a combination of results caching and index pruning for high-performance web search engines. *Proceedings of the 31st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.
- SourceForge. 2008. Open Source software development website. <http://sourceforge.net>.
- Spink A,Wolfram D,et al. 2001. Searching the web;The public and their queries. *Journal of the American Society for Information Science*,53(2):226-234.
- Stevens W R. 1996. TCP for Transactions,HTTP,NNTP,and the UNIX(R) Domain Protocols(TCP/IP Illustrated,Volume 3). New York:Addison-Wesley Pub Co.
- Stokoe C,Oakes M P,et al. 2003. Word Sense Disambiguation in Information Retrieval Revisited. *Proceedings of the Twenty-Sixth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*,SIGIR 2003. Toronto,Ont.,Canada.
- Strohman T W,Croft B. 2007. Efficient document retrieval in main memory. *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM.
- Sullivan D. 2004. Major Search Engines and Directories. <http://searchenginewatch.com/links/article.php/2156221>.
- Tellex S,Katz B,et al. 2003. Quantitative evaluation of passage retrieval algorithms for question answering. *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Toronto,Canada. ACM.
- Thota S,Carterette B. 2011. Within-Document Term-Based Index Pruning with Statistical Hypothesis Testing. *Lecture Notes in Computer Science*,6611:543-554.
- TianwangStorageFormat. 2003. Tianwang storage format of raw web pages. <http://net.pku.edu.cn/~webg/src/twformat>.
- Tomasic A,Garcia-Molina H. 1993. Performance of inverted indices in shared-nothing distributed text document information retrieval systems. *Proceedings of the 2nd International Conference on Parallel and Distributed Information Systems*. San Diego:IEEE.
- Travis B,Broder A. 2001. The need behind the query: Web search vs classic information retrieval. The 6th Search Engine Meeting. Boston,Massachusetts.
- TREC. 2006. Text REtrieval Conference(TREC). <http://trec.nist.gov>.
- Trotman A. 2003. Compressing inverted files. *Information Retrieval*,6(1):5-19.
- TSE. 2004. Homepage of Tiny Search Engine. <http://net.pku.edu.cn/~webg/src/TSE>.
- Turtle H,Flood J. 1995. Query evaluation: strategies and optimizations. *Inf. Process. Manage.*,31(6): 831-850.
- UDC. 2006. Universal Decimal Classification. <http://www.udcc.org/scheme>.
- Vivismo. 2012. Vivismo 搜索引擎. <http://www.vivismo.com>.
- Voorhees E M. 2001. Evaluation by highly relevant documents. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New Orleans,LA.
- Voorhees E M,Buckley C. 2002. The effect of topic set size on retrieval experiment error. *Proceedings of the Twenty-Fifth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*. Tampere,Finland.
- Voorhees E M,Harman D K. 2005. TREC:Experiment and Evaluation in Information Retrieval. Cambridge: The MIT Press.
- W3C. 1997. A Lexical Analyzer for HTML and Basic SGML. WWW Consortium(W3C).
- W3C. 1999. HTML 4. 01 Specification. WWW Consortium(W3C).
- Wang J,Shan S,et al. 2001. Web search engine;characteristics of user behaviors and their implication. *Science*

- in China, Series F, 44(5):351-365.
- Weiss D, Oprogramowania S I. 2001. A clustering interface for web search results in polish and english. <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.19.8549>.
- Wemble D, Jiang Y, et al. 1999. Record-boundary discovery in Web documents. Proc. SIGMOD'99.
- Wikimedia. 2008. Wikimedia. [http://commons.wikimedia.org/wiki/Main\\_Page](http://commons.wikimedia.org/wiki/Main_Page).
- Williams H E, Zobel J. 1999. Compressing integers for fast file access. Computer Journal, 42(3):193-201.
- Williams H E, Zobel J, et al. 1999. What's next-Index structures for efficient phrase querying. Proc. Australasian Database Conference. Auckland, New Zealand.
- Witten I H, Moffat A, et al. 1994. Managing Gigabytes: Compressing and Indexing Documents and Images. New York: Van Nostrand Reinhold.
- Witten I H, Moffat A, et al. 1999. Managing Gigabytes: Compressing and Indexing Documents and Images. San Francisco: Morgan Kaufmann.
- Wong C. 1997. The Robot Exclusion Standard. Web Client Programming with Perl: O'Reilly.
- Woods W A. 1973. Progress in natural language understanding; an application to lunar geology. Proceedings of the June 4-8, 1973, National Computer Conference and Exposition. New York: ACM.
- Wu L, Huang X, et al. 2004. FDUQA on TREC2004 QA Track. Proceeding of the 13th Text Retrieval Conference.
- Xie Y, O'Hallaron D. 2002. Locality in search engine queries and its implications for caching. IEEE Infocom 2002. New York, NY, United States.
- Xu J, Licuanan A, et al. 2003. TREC2003 QA at BBN: Answering Definitional Questions. Proceeding of The 12th Text Retrieval Conference.
- Yan H F, Li X M. 2002. On the Structure of Chinese Web 2002. Journal of Computer Research and Development, 39(8):958-967.
- Yan H F, Wang J Y, et al. 2001a. Architectural design and evaluation of an efficient Web-crawling system. Proceedings of 15th International Parallel and Distributed Processing Symposium. San Francisco, California, USA: IEEE Computer Society Press.
- Yan H F, Wang J Y, et al. 2001b. A dynamically reconfigurable model for a distributed Web crawling system. International Conference on Computer Networks and Mobile Computing. Beijing, China.
- Yan H, Ding S, et al. 2009. Compressing term positions in web indexes. Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval. ACM.
- Yan H, Ding S, et al. 2009. Inverted index compression and query processing with optimized document ordering. Proceedings of the 18th International Conference on World Wide Web. ACM.
- Yan H, Shi S, et al. 2010. Efficient term proximity search with term-pair indexes. Proceedings of the 19th ACM International Conference on Information and Knowledge Management. ACM.
- Yang Y. 1995. Noise reduction in a statistical approach to text categorization. Proceedings of the 18th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Seattle, WA, USA.
- Yang Y. 2001. A study on thresholding strategies for text categorization. 24th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. New Orleans, LA.
- Yang Y, Liu X. 1999. A re-examination of text categorization methods. Proceedings of ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'99).
- Yang Y, Pedersen J O. 1997. A Comparative Study on Feature Selection in Text Categorization. Proceedings of the 14th International Conference on Machine Learning (ICML'97).

- Yang Y,Zhang H. 2001. Page Analysis Based on Visual Cues. Poster Proceedings of the 10th International WWW Conference. Hongkong.
- Zamir E O. 1999. Clustering Web Documents; A phrase-Based Method for Grouping Search Engine Results, University of Washington. Doctoral Dissertation.
- Zhang D, Lee W S. 2003. Question classification using support vector machines. Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval. Toronto, Canada; ACM.
- Zhang F, Shi S, et al. 2010. Revisiting globally sorted indexes for efficient document retrieval. Proceedings of the 3rd ACM International Conference on Web Search and Data Mining. ACM.
- Zhang J, Long X, et al. 2008. Performance of compressed inverted list caching in search engines. Proceeding of the 17th International Conference on World Wide Web. ACM.
- Zhang Z, Chen J, et al. 2004. A preprocessing framework and approach for web applications. Journal of Web Engineering, 2(3):175-191.
- Zheng L, Cox I. 2009. Entropy-based static index pruning. Advances in Information Retrieval, 34(2):713-718.
- Zheng L, Cox I J. 2009. Document-oriented pruning of the inverted index in information retrieval systems. Advanced Information Networking and Applications Workshops, 697-702.
- Zhu M, Shi S, et al. 2007. Effective top-k computation in retrieving structured documents with term-proximity support. Proceedings of the 16th ACM Conference on Information and Knowledge Management. ACM.
- Zhu M, Shi S, et al. 2008. Can phrase indexing help to process non-phrase queries? Proceeding of the 17th ACM Conference on Information and Knowledge Management. ACM.
- Zobel J. 1998. How reliable are the results of large-scale information retrieval experiments? Proceedings of the 1998 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval(SIGIR'98). Melbourne, Vic., Aust.
- Zobel J, Moffat A. 2006. Inverted files for text search engines. ACM Comput. Surv., 38(2).
- Zukowski M, Heman S, et al. 2006. Super-Scalar RAM-CPU Cache Compression. 22nd International Conference on Data Engineering(ICDE'06). IEEE.

## 附录 术 语

### A

**暗网数据**(Deep Web Data),一般指搜索引擎无法抓取到的信息,主要由两类信息构成:一类是由于技术实现的原因无法抓取,如很多网站本身不符合协议规范,导致搜索引擎的爬虫无法识别这些网站的内容并抓取;另一类是网站提供的信息是存储在网络数据库中的内容,搜索引擎难以通过网页抓取方式获取其全部信息内容。

### B

**半结构化数据**(Semi-Structured Data),和普通纯文本相比,Web 上的网页数据具有一定的结构性,表现在其中的 HTML 标注上;但和具有严格理论模型的关系数据库的数据相比,这种 HTML 标注带来的结构性又要弱很多,因此人们称 Web 上的数据为半结构化数据,这是 Web 上数据的基本特点。

**标签云**(Tag Cloud),以标签的形式来表示一个网站中的内容。这些标签一般依照字典排序,按热门程度确定字体的大小和颜色。这样,依照字典或者热门程度来寻找信息便成为可能。每一个标签通常是一个超链接,指向分类页面。

**编辑距离**(Edit Distance),又称 Levenshtein 距离,是指两个字符串之间,由一个转成另一个所需的最少编辑操作次数,其中一次编辑操作是指将一个字符替换成另一个字符、插入一个字符,或者删除一个字符等。

**布尔模型**(Boolean Model),在信息检索领域,不同的场合有不同的含义。当讨论用户提交查询的时候,指的是为形成最终查询结果集合,由一个查询的各个成分对查询结果子集之间所要求的一种运算关系;而在讨论文档比较的向量空间模型中,布尔模型指的是构成一个文档向量的各个分量只取 1 和 0 两个值,分别代表对应特征项的出现与否。

### C

**查全率**(Recall),判断检索系统质量的一种度量,表示系统所检索到的与查询相关的文档数占与查询相关的总文档数的百分比。

**查询**(Query),用户使用信息系统提供的输入语言和规则对自己信息需求的一种表达。常用的输入语言包含关键词规范和一些布尔连接符。

**查询扩展**(Query Expansion),在原用户查询词的基础上,通过一定的方法和策略把与原查询词相关的词、词组添加到原查询中,组成新的、更能准确表达用户查询

意图的查询词序列,然后用新查询对文档重新检索,从而提高信息检索中的查全率和查准率。

**查询延迟(Query Latency)**,在用户提交一个查询与系统获得返回结果之间用户需要等待的时间。在测算时,通常以毫秒或秒为单位进行计算。

**查准率(Precision)**,判断检索系统质量的一种度量。系统所检索到的与查询相关的文档数占检索出的所有文档数的百分比,即反映检索结果“正确性”的度量。

**词典(Vocabulary)**,文档(或文档集合)中所有不同词项的集合。

**词频(Term Frequency, tf 或 TF)**, $TF(i, j)$ 是指一个词项  $t_i$  在一篇文档  $d_j$  中出现的次数。

## D

**代理(Agent)**,或称代理程序,在应用中,接收到用户的请求后,能代表用户完成任务并返回结果,但不受用户监督的程序、进程或部分系统。在应用中,代理程序用于从存档或信息库中搜索与用户所给主题词相关的内容,所以有时又称为智能代理(Intelligent Agent)。

**倒排文件(Inverted File)**,组织和索引文件,以便于检索的一种方法。在该方法中,一个关键字的集合是基础,该集合中每一个关键字对应一串记录项,其中每一项包含一个文档编号、该关键字在该文档中出现的情况等信息。

**倒置文档频率(Inversed Document Frequency, IDF)**,通常  $IDF(t_i)$  取值为  $\log(N/n_i)$ ,其中  $N$  是所有文档的总数, $n_i$  是在  $N$  个文档中包含词项  $t_i$  的文档数。

**动态网页(Dynamic Web Page)**,需要通过提交查询信息才能获取的网页。

**动态摘要(Dynamic Abstract)**,做文档摘要的一种方法。对于搜索引擎来说,就是在响应用户查询的时候,根据查询词在文档中出现的位置,提取出查询词周围相关的文字并返回给用户。由于一篇文档会含有不同的查询词,因此动态摘要技术可能把同一个文档形成不同的摘要文字。

**多媒体信息检索(Multimedia Information Retrieval)**,将多媒体信息按一定的方式进行特征提取,并组织起来,根据用户的需求找出有关信息的过程,主要包括对声音、图形、图像、动画、视频等信息的检索。

## G

**共有词汇假设(Shared Bag of Words)**,信息检索技术的一个最基本假设,即认为文档的含义可以由它所包含的关键词的集合来表达。

**共指消解(Co-reference Resolution)**,将现实世界中同一实体的不同描述合并到一起的过程,是自然语言处理、机器翻译、信息抽取等领域的主要技术之一。“共指”是指文本中不同的语言元素指称同一对象的现象。

**关键词堆砌(Keyword Stuffing)**,在网页中大量堆砌关键词,从而提高关键词密



度,使搜索引擎抓取到该网页的概率增大。常见的关键词堆砌形式有在标题中堆砌关键词、在网页中堆砌关键词以及隐藏关键词堆砌。

## H

**HTML**(HyperText Markup Language,超文本标记语言)是 Web 的关键技术之一,它为 ASCII 格式的超本文档提供了一种标准表述方式。

**缓存**(Cache),在计算机科学领域经常出现的一个概念,其基本含义是利用局部性原理实现的一种匹配两种不同速度的中间机制。它可以出现在 CPU 和 RAM 中间,也可以出现在应用系统的 I/O 操作与磁盘之间。在搜索引擎中,为缓解查询要求的高速度和磁盘访问低速的矛盾,常会在内存中设计各种缓存,包括查询缓存、点击缓存,以及倒排表缓存等。

## J

**基于内容检索**(Content-Based Retrieval),一种针对多媒体信息的检索方式。指通过借助模式识别、语音识别、图像理解等相关领域的研究成果,对多媒体数据的听、视觉特征和语义特征进行自动(或半自动)的分析、表达和组织,从而对多媒体对象及其蕴含的内容和语义特征进行检索的一种检索方式。

**交叉语言检索**(Cross Language Information Retrieval,CLIR),允许用户以它们熟悉的语言构造检索提问式,然后使用该提问式检索以系统支持的其他语言写成的文献。在 CLIR 系统的匹配过程中,跨越语言障碍的策略主要包括同源匹配策略、提问式翻译策略、文献翻译策略和语言转换技术等。

**竞价排名**(Paid Listing),商业搜索引擎提供的一项按效果付费的服务。企业的产品以关键词的形式在搜索引擎平台上作推广,当企业购买该项服务后,通过注册一定数量的关键词,其推广信息就会率先出现在网民相应的搜索结果中。

**静态网页**(Static Web Page),不需要通过提交查询信息即可获得页面。

**镜像网页**(Mirror Web Page),网页的内容完全相同,未加任何修改。

**镜像网站**(Mirror Site),将一个网站源程序放到几个服务器中,从而形成几个相同的网站。这些网站分别拥有自己独立的 URL,内容则与主站没有太大差别。

**局部性原则**(Locality Principle),是程序行为的一种性质。它包括时间局部性和空间局部性。前者指的是,如果某数据刚才被访问,则它很可能在近期内还要被访问;后者指的是,如果某数据刚才被访问,则和它在位置上相邻的数据很可能将被访问。

**拒绝服务攻击**(Denial of Service,DoS),是一种攻击行动,使网站服务器充斥大量要求回复的信息,消耗网络带宽或系统资源,导致网络或系统不胜负荷以至于瘫痪而停止提供正常的网络服务。

## L

**链接分析**(Link Analysis): Web 上的网页及其相互之间的链接可以看成是一个巨大的有向图, 链接分析指的是利用网页之间的链接信息来评判其重要性(或者相关性)的技术。常用的链接信息包含网页的出度、入度, 锚文本内容等; 常用的链接分析算法有: PageRank, HITS, SALSA, PHITS, Bayesian 等。

**链接作弊**(Link Spam), 通过构造一系列迷惑搜索引擎的链接结构, 使某些 Web 页面在检索结果中的排名高于实际排名的行为。常见策略包括蜜罐诱饵、渗入 Web 目录、链接交换、购买过期域名、构造链接工厂等。

## M

**MD5**(Message Digest 5), 报文摘要, 用于报文编码的一种算法。MD5 算法在 RFC1321 中定义, 其基本功能是将一个任意长的报文变换为一个 128 位的摘要, 两个不同的报文对应的摘要相同的概率极小, 两个摘要之间的相近程度和对应两个报文的相近程度没有任何关系。

**锚文本**(Anchor Text), HTML 文本中的链接描述信息, 向读者提示该链接所指向网页的性质或特征。例如, 在一篇网页中书写有 `<a href = "http://www.cctv.com">新闻频道</a>`, 则“新闻频道”就是链接 `href = "http://www.cctv.com"` 在本网页中的锚文本。

**命名实体识别**(Named Entity Recognition, NER), 识别文本中具有特定意义的实体, 该过程是信息提取、问答系统、句法分析、机器翻译等应用领域的重要基础工具。其识别对象包括三大类(实体类、时间类、数字类)和七小类(人名、机构名、地名、时间、日期、货币、百分比)命名实体。

**目录型网页**(Hub Page), 该网页提供很多指向其他权威型网页的超链接。是与权威型网页相对应的。

## P

**PageRank 算法**(PageRank Algorithms), 用于标识网页的等级或重要性的一种方法, 最早应用于 Google 搜索引擎系统网页权值的计算。它根据一个网页外部链接和内部链接的数量和质量来衡量该网页的价值。

## Q

**齐普夫定律**(Zipf's law), 由美国学者 G. K. 齐普夫于 20 世纪 40 年代提出的词频分布定律。它可以表述为: 如果把一篇较长文档中每个词出现的频次统计起来, 按照高频词在前、低频词在后的递减顺序排列, 并用自然数给这些词编上等级序号, 即频次最高的词等级为 1, 频次次之的等级为 2, ……若用  $f$  表示频次,  $r$  表示等级序

号,则有  $f=C/r$  ( $C$  为常数)。

**切词**(Word Segmentation),或称分词,主要在中文信息处理中使用,即把一句话分成一个词的序列。如“网络与分布式系统实验室”,分词为“网络与分布式系统实验室”。

**全文检索**(Full Text Retrieval),文本信息检索的一种方法(或者说是一种精细程度),其特点是不仅文档中出现的每一个词都可以被检索出来,而且每一个词的每一次出现也可以被检索出来。

**权威型网页**(Authority Page),网页内容通常有一个特定的主题,并且被许多其他网页链接,是与目录型网页相对应的一个概念。

## S

**散列表**(Hash Table),或称哈希表,是一种数据结构,它便于快速的信息查找。散列表生成时为表中的每项数据分配一个随机索引代码。这种索引代码的随机性使得数据的分布比较均匀,从而可能大大节省后续查找的时间。

**社会化搜索**(Social Search),搜索结果考虑用户的交互性和贡献的一种网站搜索方法。社会化搜索通常考虑社会化因素,例如交互、联系、用户行为模式等。通过社会化搜索,可形成有共同爱好的人际圈子,最终达到一个全社会知识共享的目的。

**数字图书馆**(Digital Library),一个数字信息对象收藏、组织和表现这些对象的方法以及将这些对象提供给用户的相关的信息技术。它包括支持用户进行定位、检索和获取这些信息对象的服务。

**受控词**(Control Term),一种经过规范化处理的检索词汇,一般取自于标题表、叙词表或分类表等。

**搜索引擎**(Search Engine, SE), Web 上的一种应用软件系统,它以一定的策略在 Web 上搜集和发现信息,对信息进行处理和组织后,为用户提供 Web 信息查询服务。

**搜索引擎优化**(Search Engine Optimization, SEO),利用搜索引擎工作原理提高网页(或网站)排名的技术与方法。SEO 通过了解各类搜索引擎如何抓取互联网页面、如何进行索引以及如何对某一特定关键词的搜索结果进行排名等技术,对网页进行相关的优化,从而提高搜索引擎排名,增加网站访问量,最终提升网站的销售能力或宣传能力。可分为站内搜索引擎优化和站外搜索引擎优化。

**索引词载体信息**(Index Term Carrier),HTML 的标签信息标识了文档中索引词的字体和大小写等信息。

## T

**停用词**(Stop Word),指文档中出现的连词、介词、冠词等并无太大意义词。例如,在英文中常用的停用词有 the, a, it 等;在中文中常见的有“是”、“的”、“地”等。

**吞吐量**(Throughput),或称吞吐率,是指在单位时间里系统完成的总任务量。对于搜索引擎来说,就是指系统在单位时间(秒)里可以服务的最大用户查询数量。

## U

**URL**(Uniform Resource Locator),用来定位互联网上信息资源的一种协议(或者说描述规范),网页的定位通常就是以形如“http://host/path/file.html”的 URL 来描述的,而 FTP 资源则用形如“ftp://host/path/file”的 URL 来描述。

**URL 重定向**(URL Redirection),当使用者浏览某个网址时,将其导向另一个网址的技术。常见的 URL 重定向方式包括 301(永久重定向)与 302(暂时重定向)两种。

**URL 目录深度**,网页对应的 url 中除去域名部分的目录层次,即 url = schema://host/localpath 中的 localpath 部分。例如, url 为 http://www.pku.edu.cn, 则目录深度为 0;如果是 http://www.pku.edu.cn/cs,则目录深度为 1。

**URL 域名深度**,网页对应的 url 中域名部分包含的子域个数。

## W

**网页出度**(Page Outdegree),针对一个网页,该网页指向其他网页的超链接数目。

**网页净化**(Noise Reduction),识别并去除网页噪音的过程;即去除网页内与该网页主题内容无关的信息,如广告、版权信息等。

**网页爬取器**(Gatherer),指网页搜集子系统中根据 url 完成一篇网页抓取的进程或者线程,通常一个搜集子系统中会同时启动多个 gatherer 并行工作。

**网页入度**(Page Indegree),针对一个网页,整个网络中指向该网页的超链接数目。

**网页搜集子系统**(Crawler System),尤指在搜索引擎系统中,利用 HTML 文档之间的链接关系,在 Web 上依照网页之间的超链关系一个个抓取网页的程序。鉴于其在 Web 上沿超链“爬行”的工作方式,这种程序有时也称为“蜘蛛”(spider)。Crawler, spider, robot, bot 一般都指的是相同的事物。

**文档对象模型**(Document Object Model, DOM),DOM 将一个 XML 文档转换成一个对象集合,然后可以任意处理该对象模型。这一机制也称为“随机访问”协议,因为可以在任何时间访问数据的任何一部分,然后修改、删除或插入新数据。

**文档自动分类**(Automatic Text Categorization, ATC),用计算机程序来确定指定文档和预先定义文档类别之间的隶属关系。

**文本过滤**(Text Filtering),根据一定的标准和要求,从文本信息流中选取用户相关的信息或删除不相干信息的过程和方法。

**文本检索会议**(Text Retrieval Conference, TREC),该会议负责组织收集并向与

会者提供标准的语料库、检索条件和问题集以及评测办法,与会者则被要求在规定的时间内构造检索系统并提交检索结果,由会议负责评测各个检索结果的优劣,最终依据评测结果召开大会进行学术交流,发表会议论文。

## X

**XML 检索(XML Retrieval)**,一种基于 XML 语言的结构化特性而进行的检索。由于 XML 语言可以将文档分成许多部件并对这些部件加以标识,因此在 XML 检索中,文档中各层次的 XML 元素都是可检索的单元,从而使检索系统更关注文档中的结构和粒度问题,有助于实现“内容+结构”的检索。

**先进先出(First In First Out, FIFO)**,是一种页面替换算法,选择最先装入主存储器的那一页调出,或者说是把驻留在主存时间最长的那一页调出。

**相关反馈(Relevance Feedback)**,检索系统中提供的一种人机交互机制。在这种机制的作用和控制下,系统可以根据初试检索结果的相关性判断,对用户的检索要求(或提问式)进行适当的修改和扩展,并据此进行检索,从而得到改进的查询结果。实现相关反馈的方式主要有两种:一种是需要检索用户主动配合和参与的方式(用户反馈法),另一种则不需要用户的参与,由系统通过自动分析技术来完成(系统自动反馈法)。

**相关排序(Relevance Ranking)**,指信息检索系统返回结果的排序,其中条目的顺序反映了系统确定的结果和查询的相关程度。

**向量空间模型(Vector Space Model, VSM)**,按照共有词汇假设,一组文档有一个总词语集合  $\Sigma$ ,一篇文档可以用一个向量表示,其元素是对应词语在该文档中出现情况的一种定量描述,一组文档就可以看成是一个向量空间中的若干元素,于是可以应用向量空间中距离的概念来考察两篇文档之间的相似程度等。

**响应时间(Response Time)**,在计算机系统中,从提交请求(或询问)到开始看到回答之间所经历的时间。对于搜索引擎来说,就是用户提交查询到他看到返回结果之间所经历的时间。在搜索引擎的具体实践中,由于这个时间和动态变化的网络状态有关,常常用检索系统为完成一个查询所消耗的响应时间来近似。

**消重(Replicas or Near-replicas Detection)**,清除所搜集网页集合中的镜像或转载网页的过程。

**协议(Protocol)**,为实现通信而制定的能够协调各功能单元操作的一组规则。

**信息检索(Information Retrieval, IR)**,将信息按一定的方式组织和存储起来,并根据用户的需要找出有关信息的过程。

**信息检索模型(IR model)**,依照用户查询,对文档集合进行相关排序的一组前提假设和算法。IR 模型可形式地表示为一个四元组  $\langle D, Q, F, R(q_i, d_j) \rangle$ ,其中  $D$  是一个文档集合,  $Q$  是一个查询集合,  $F$  是一个对文档和查询建模的框架,  $R(q_i, d_j)$  是一个排序函数,它给查询  $q_i$  和文档  $d_j$  之间的相关度赋予一个排序值。常用的信息

检索模型有集合论模型、代数模型、概率模型等。

## Y

**用户查询日志(User Query Log)**,是在用户提交查询请求时由系统自动记录的相关信息,它包括用户查询时提交的关键词、提交时间、用户 IP 地址、页号(通常查询结果分页显示,每页显示 10 个查询结果,用户首次查询页号为 1,用户翻页时页号即为用户选择的结果页面号)和是否在缓存中命中等信息。

**用户点击日志(User Hit Log)**,是用户浏览查询结果并点击页面时由系统自动记录的相关信息,它通常包括用户点击页面的时间、点击页面的 URL、用户 IP 地址、点击页面的序号(该页面在查询结果中的位置)、该点击对应的查询词等信息。

**用户体验(User Experience, UE)**,指人们针对使用或期望使用的产品、系统或者服务的认知印象和回应,包括情感、信仰、喜好、认知印象、生理和心理反应、行为和成就等各个方面。

**元数据(Meta Data)**,描述某种类型资源(或对象)的属性、并对这种资源进行定位和管理、同时有助于数据检索的数据。

**语义网(Semantic Web)**,一种能够根据语义进行判断的网络。在这样的网络中,信息都被赋予了明确的含义,机器能够自动地处理和集成该网络中的可用信息。语义网使用 XML 定义定制的标签格式,用 RDF 灵活的表达数据,并通过本体描述语言 OWL(Ontology Web Language)来描述网络中术语的明确含义和它们之间的关系。

**元搜索引擎(Meta Search Engine)**,又称集成型搜索引擎,它将用户的查询发送给多个独立的搜索引擎,收集它们产生的结果,然后按照一定的算法进行选择 and 重新排序以形成一个最终结果返回给用户。

## Z

**在线社区(Online Community)**,指处于在线环境中进行交互且具有共同的目的、特性或兴趣的实体群组。

**中文信息处理(Chinese Information Processing)**,用计算机对汉语的音、形、义等语言文字进行信息的加工和操作,包括对字、词、短语、句、篇章的输入、输出、识别、转换、压缩、存储、检索、分析、理解和生成等各方面的处理技术。

**主题搜集(Topic-Specific/Focused Crawling)**,即面向主题的信息搜集系统,其主要任务是利用有限的网络带宽、存储容量和较少的时间,抓取尽可能多的与主题内容密切相关的网页。

**转载网页(Near-replicas Web Page)**,内容基本相同但可能有一些额外的编辑信息等。虽然网页做了部分改动,但其主题内容未变,即去除网页的噪声(如广告、版权等信息)外,其他正文内容相同。转载网页也称为近似镜像网页。

**自动问答系统**(Automatic Answering System),指综合运用知识表示、信息检索、自然语言处理等技术,能够识别用户以自然语言(而不是关键词组合)输入的问题并返回简洁、准确答案的检索系统。

**自动文摘**(Automatic Abstract),利用计算机技术自动从原始文献中提取出可全面准确地反映某一文献中心内容的简单连贯的短文,主要包括基于统计的自动文摘、基于理解的自动文摘、基于信息抽取的自动文摘以及基于结构的自动文摘等。

**最低频使用**(Least Frequently Used, LFU),缓存内容维护的一种数据替换策略,当缓存满,且有新的数据要进来时,它总是淘汰现有数据中在过去使用频率最低的数据。数据替换的粒度可以根据应用场合确定。

**最近最少使用**(Least Recently Used, LRU),缓存内容维护的一种数据替换策略,当缓存满,且有新的数据要进来时,它总是淘汰现有数据中在过去最长时间未被使用过的数据。